deJovi Computer Simulator

CPSC-300 - Software Engineering Fall 2013

Course Instructor: Dr Desanka Polajnar

Project Team:

- Devon Harker
- Josh Haskins
- Vincent Tennant

December 5, 2013

Table of Contents

1. Introduction

1.1. Purpose of Document

The purpose of this document is to outline and describe the exact details of what this project will become. This document will also aid us in actually designing the software for the end user.

1.2. Glossary (Definitions, Acronyms and Abbreviations)

1.2.1. **Simulation** - Is the act of running a user supplied program in simulated computer environment, while also displaying the internal details of the processor execution.

1.2.2. **Manager** - The main "engine" for the simulator. The Manager ensures that the simulation runs as expected and controls the flow of execution, and information and data within the system.

1.2.3. **Memory** -_A virtual storage unit equivalent to RAM in a physical computer. This is where programs are stored.

1.2.4. **Registers -** A small amount of storage that is part of the digital circuitry within a computers' CPU

1.2.4.1. **AC** – Accumulator Register, location where intermediate arithmetic and logic results are stored.

1.2.4.2. **PC** – Program Counter, holds the address to the next instruction to be executed.

1.2.4.3. **IR** – Instruction Register, stores the instruction that is currently being decoded or executed.

1.2.4.4. **MR** – Memory Register, holds the data that is to be written to memory, or the data that has been read from memory.

1.2.4.5. **MAR** – Memory Address Register, stores the memory address for either the data that will be loaded and given to the CPU or that will be written to memory

1.2.5. **ALU** – Arithmetic Logic Unit, "performs integer arithmetic and logical operations" [2].

1.2.6. **CU** – Directs the operation of the processor.

1.2.7. **GUI** - Graphic User Interface, this is the interface in which the user controls the simulation through and is able to view simulation progress and statistics.

1.2.8. **Simulation States**

1.2.8.1. **Ready -** The initial state for the simulation. If the simulator has just been started, register values and memory will be 0. If a simulation has just been running or the user has modified register and/or memory values, they may not be 0. The user is able to modify register and memory values. The user can save the entire current simulation,which includes the users program, memory and register values; the user may also choose to save just the simulation memory. The user can load a previous simulation from a save file. The user is able to reset the values stored in the registers and memory to 0. The user may start a simulation which will then proceed to the Running State. The user can enter the Paused state by choosing to step through a single simulation cycle.

1.2.8.2. **Running** - This is when the simulation is executing the users program stored in memory. The user can choose to reset, which will reset the memory and register values to 0 and return to the Ready state. The user can also stop execution at any time. When a user stops a simulation the simulation immediately returns to the Ready state and the PC is reset to its default value 0. Simulations can be paused, when running, at any time. When paused the simulation immediate goes to the Paused state, from there more actions can be performed. If all goes well (ie. no syntax or user program errors), the simulation will compete itself return to the ready state, leaving all register and memory values unchanged.

1.2.8.3. **Paused** - When a simulation is in the Paused state the actor has the option of Resetting the simulation and stopping the simulation as they do in the Running (1.2.7.2) and Ready state. Again, as in the Ready (1.2.7.1) state the actor may Save All or Save Memory for later use. The actor also has the option of stepping through individual simulation cycles. This allows the actor to view exactly what is occurring within the simulated computer. If the user steps through the simulation and the final instruction is executed, the simulation immediately returns to the ready state, leaving the register and memory values unchanged.

1.3. References

[1] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed. Boston: Prentice Hall, 2006.

[2] Wikipedia contributors. (2013, Nov 18). *Arithmetic Logic Unit*. [Online]. Available: http://en.wikipedia.org/wiki/Arithmetic_Logic_Unit

[3] Michael R. Blaha and James R Rumbaugh, *Object-Oriented Modeling and Design with UML*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2004.

[4] Daniel P. Friedman and Mitchell Wand, *Essentials of Programming Languages*, 3rd ed. Cambridge, MA: The MIT Press, 2008.

[5] Wikipedia contributors. (2013, Nov 19). *Central Processing Unit*. [Online]. Available: http://en.wikipedia.org/wiki/Cpu

2. System Overview

2.1. Problem Statement

A high school computer science course is in need of a computer processor simulator. The main purpose of the simulator is to introduce students to the principles of computer architecture and organization. The simulator must be able to run programs in a simple machine language while displaying the internal processor details of their execution.

The simulated computer consists of a single Central Processing Unit (CPU) and Internal Memory (MEM), connected by a 16-bit Data Bus and 14-bit Address Bus. Its instruction and data words are 16 bits wide. Each CPU instruction has a single memory operand. The instruction format consists of a 2-bit operation code and 14-bit memory address.

The CPU consists of: Accumulator Register (AC), Program Counter (PC), Instruction Register (IR), Memory Register (MR), Memory Address Register (MAR), the Arithmetic- Logic Unit (ALU) and the Control Unit (CU). All registers are 16-bit wide, except MAR which is 14 bits wide. Integer values are represented in 2's complement format.

The simulator should have a graphical user interface that enables users to enter simple (possibly single-instruction) programs and monitor their execution. The user should be able to START the simulation, RUN to the end of program execution, STEP through cycles of instruction execution, STOP the simulation, RESTART the simulation from the stopping point, and RESET the computer to its initial state from any present state. The content of all CPU registers, the internal memory content, as well as the values of all control signals generated by the CU should be displayed to the user. In addition, the CU state machine should be displayed with highlighted active states during instruction execution.

2.2. System Features - Requirements Definition

2.2.1. Simulation Control Functions

2.2.1.1. **Pause** - This temporarily pauses the simulation, allowing the user to view exactly what values are being held in registers and what actions are currently being made by the rest of the system. When the system is paused the user has the ability to step through the execution of the program.

2.2.1.2. **Run** - Executes the program from start to finish, unless interrupted by the Pause or Stop command. It begins at the initial PC value of 0. In MEM[0] the location of the loaded program is stored in a JMP command to tell the PC where to go for the execution.

2.2.1.3. **Step** - This allows the user to step though instruction execution one instruction at a time. Every time the user clicks the Step button the simulation is allowed to progress with one more instruction. If the last instruction is executed, the simulation is returned to the Ready state, with the PC counter reset back to 0.

2.2.1.4. **Resume** - This resumes the simulation from a Paused state. Execution continues at the preset rate.

2.2.1.5. **Reset** - Sets the AC, PC, IR, MR, MAR, and all memory locations to a value of zero. A simulator that has just been started and a simulator that has just been reset are identical.

2.2.1.6. **Load Simulation** - Loads a file from the users system. This file contains the necessary information for the program to resume or begin execution from a previous program and simulation state.

2.2.1.7. **Save Memory** - Saves the current contents of the simulators memory to an external file. This file can be loaded on a later date to allow the user to load a program or program date whenever needed.

2.2.1.8. **Save All** - Saves the simulations current state as well as the simulations current memory contents. All of this is saved into an external file to allow the user to load a previous program as well as that programs current simulation state.

2.2.1.9. **Simulation Speed** - Allows the user to adjust the speed of the simulation. On the fastest setting, the value of the registers and the state of the control unit will update extremely quickly, too fast for the user to read. On the slowest setting, the user will be able to see the registers being updated and they can see the control unit in its different states.

2.2.2. Input Functions

2.2.2.1. **Load Program Into Memory** - Allows the user to load the program into a specific memory location using the displayed text window. Sets MEM[0] to ensure that the PC jumps to the location of the program for proper execution.

2.2.2.2. **Program Adjustment** - User enters the memory address to adjust, then the value to be stored in that address.

2.2.2.3. **Register Adjustment** - When the program is stopped, paused or waiting to be executed the user can manually adjust the register values before or during execution. The user will select the register to change, input the value to change it and the register will change.

3. System Architecture

3.1. Architecture Diagram

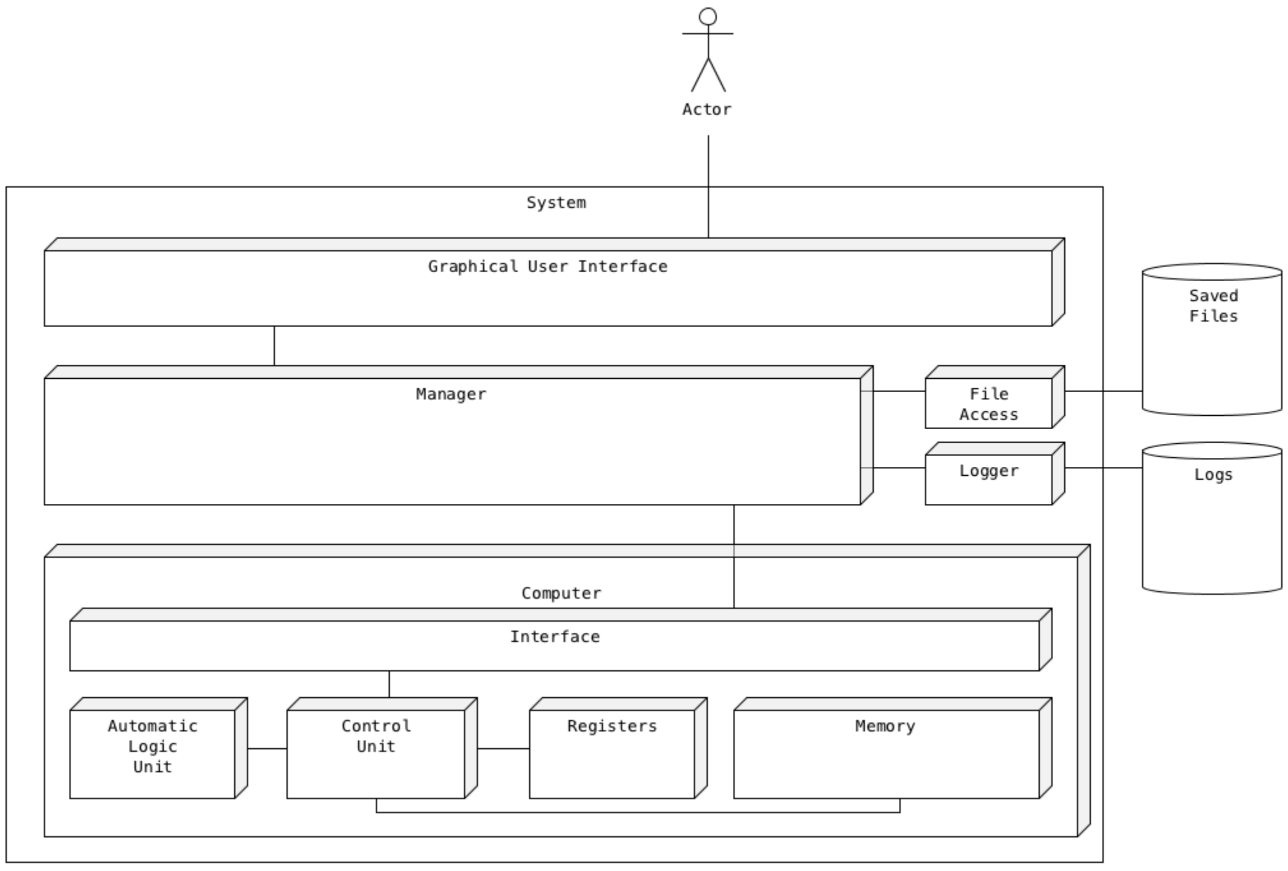


Figure 1. The Architecture Diagram of our system.

This is the basic system architecture for out computer simulator. It showcases the different components within our system, how they communicate and work together. The Actor(be it Student or Teacher) interacts with the graphical user interface. The GUI provides the user with the information needed to create and run a simulation as well as acting as the main source on input for the system itself. the Manager(as defined in 1.2.2) acts as the main controller for the system. The Manager provides the GUI with information to display and takes information from the GUI, provided by the user, to use in the simulation. The Manager acts as the main controller for the entire system, controlling components such as the GUI by telling it what to display and when, and File Access by telling it what files to load and save. File Access is the component that performs the saving and loading of external simulation files. Logger keeps a log of all events that occur during each individual execution of the System. The simulated Computer acts as one larger component made up of smaller components. The interface allows the System to access the Computer easily, Control Unit acts as the main controller for the Computer(CPU), The Arithmetic Logic Unit performs basic mathematical operations, Registers simulate the Registers within the Computers CPU, and finally the memory is the Main Memory for the computer itself. User programs and user data from those programs are stored here as they would be in a physical computer.

4. System Requirements Specification

4.1. Use Case Diagram

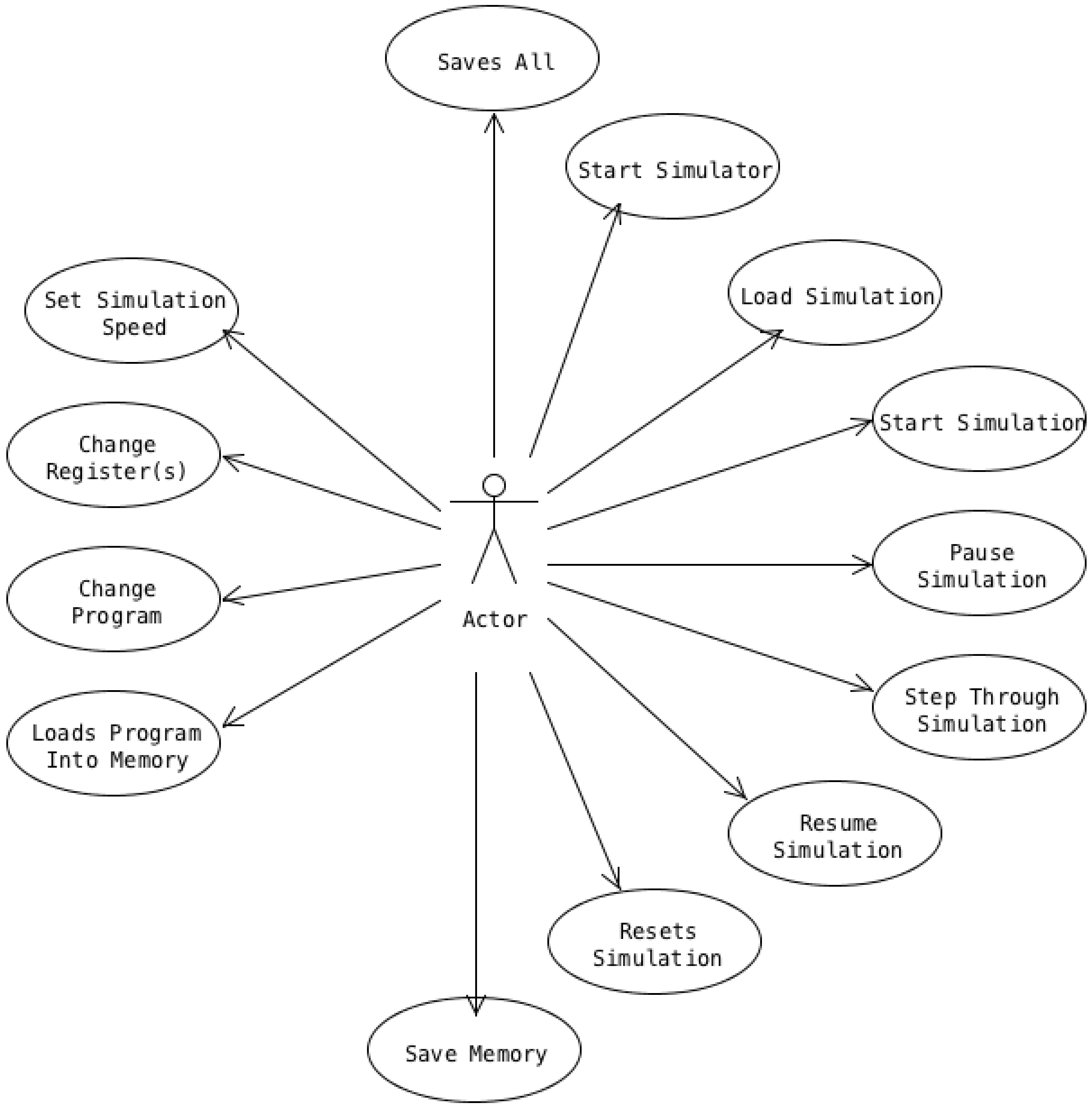


Figure 2. The Use Case Diagram of our system.

Figure 2 displays the uses cases in our system. It showcases all of the functions that the user is presented with when using the simulator.

4.2. Use Cases

4.2.1.1. Save All

Summary: All of the simulation information is saved to an external file.

Actor: Student, Teacher

Preconditions: Simulation is in the Ready or Paused states.

Description: The user wants to save their program to a file they have to option to do so even when no program has been input. When the user requests 'Save Program'. The system will ask for the desired name of the file to save to. Once ready, the Simulation will save the users program to the default save location(the directory of the main .jar file). If the simulation is paused the the user has the option of saving the simulation in the paused state.

Exceptions: *Null file:* The user did not enter a file or file name. The System informs the user of the error and prompts the user again to select a file.

Cancel: The user selects 'Cancel'. The System aborts the 'Save All' request.

Post-conditions: The simulation data is successfully saved to the selected file.

4.2.1.2. Reset Simulation

Summary: All register values and the simulation memory is cleared and set the default values.

Actor: Student, Teacher

Preconditions: Simulation is in the Ready or Paused states.

Description: The user requests 'Reset'. The system will then ask if the user if they would like to save. If the user selects 'Yes', the system will prompt the user for the desired name of the file to save to. The file is saved in the default save file location. Next the registers and the memory are all cleared and are set to their default value of 0.

Exceptions: None.

Post-conditions: The memory and registers are set to the default values of 0.

4.2.1.3. Pause Simulation

Summary: The simulation is paused.

Actor: Student, Teacher

Preconditions: Simulation is in the Running state.

Description: The user requests 'Pause'. The system then pauses the simulation. While in the paused state the user has options to perform additional tasks, such as step through each cycle. When the user is ready to exit the Paused state, they may request Resume or they can exit the simulator.

Exceptions: None.

Post-conditions: The simulation of the program loaded into memory.

4.2.1.4. Run Simulation

Summary: Begins simulating user code.

Actor: Student, Teacher

Preconditions: Simulation is in Ready state and a program is loaded into memory.

Description: The user requests 'Run' and the System starts simulating the program that is currently in memory. Once the simulation has been started, the user is presented with a few options during execution. These options are, Pause and Stop.

Exceptions: *Runtime exception:* An instruction could not simulated correctly.

Post-conditions: The program loaded in memory is now being simulated.

4.2.1.5. Load Program Into Local Memory

Summary: Loads a user program into simulation's memory.

Actor: Student, Teacher

Preconditions: Simulation is in Ready state, the contents to be loaded is formatted correctly.

Description: The user requests 'Load Program Into Local Memory'. The System checks the program and determines if it is syntactically correct. If the program is syntactically correct it is then injected into the memory at the location that is designated for holding programs, overwriting any previous programs.

Exceptions: *Syntax error:* An instruction in the program is not syntactically correct. The System informs the user and aborts loading the program into memory.

Post-conditions: The program has been loaded into the simulator's memory.

4.2.1.6. Load Simulation

Summary: Loads a previously saved simulation into the simulator.

Actor: Student, Teacher

Preconditions: Simulation is in Ready state, the file to be loaded already exists.

Description: The requests 'Load Simulation'. The system will then prompt the user to chose a file. The contents are copied to the GUI, where it can be edited by the user or loaded into memory with the use of the 'Load Program into Memory'.

Exceptions: *Null file:* The user has not entered a file or file name. The System informs the user of the error and prompts the user again to select a file.

Bad file: The file the user has selected is not of the right extension or it is of the wrong format. The System informs the user of the error and prompts the user again to select a file.

Post-conditions: The contents of the file are copied to the user interface.

5. System Models

5.1. Scenarios

5.1.1. Save Program to File

User requests 'Save Program to File'
System asks User for desired file name
User enters a file name or selects a pre-existing file
System saves simulation essential information in the file
System confirms saved file

5.1.2. Reset Simulation

User requests 'Reset'
System confirms simulation reset
User gives confirmation
System resets memory and register values to zero

5.1.3. Pause Simulation

User requests 'Pause'
System pauses simulation

5.1.4. Start Simulator

User starts Simulator
System displays user interface

5.1.5. Quit Simulator

User requests simulator to quit
System confirms and quits

5.1.6. Run Simulation

User requests simulation run
System runs simulation
System displays results of simulation to User

5.1.7. Load Program Into Local Memory

User requests “Load Program into Local Memory”
System loads program into memory
System confirms program is loaded

5.1.8. Load Program From File

User requests ‘Load Program From File’
System opens file browser
User selects desired file
System opens selected file and changes values accordingly
User makes desired changes, if any

5.2. Sequence Diagrams

See Scenarios (5.1) for a detailed explanation of each diagram.

5.2.1. Save Program to File

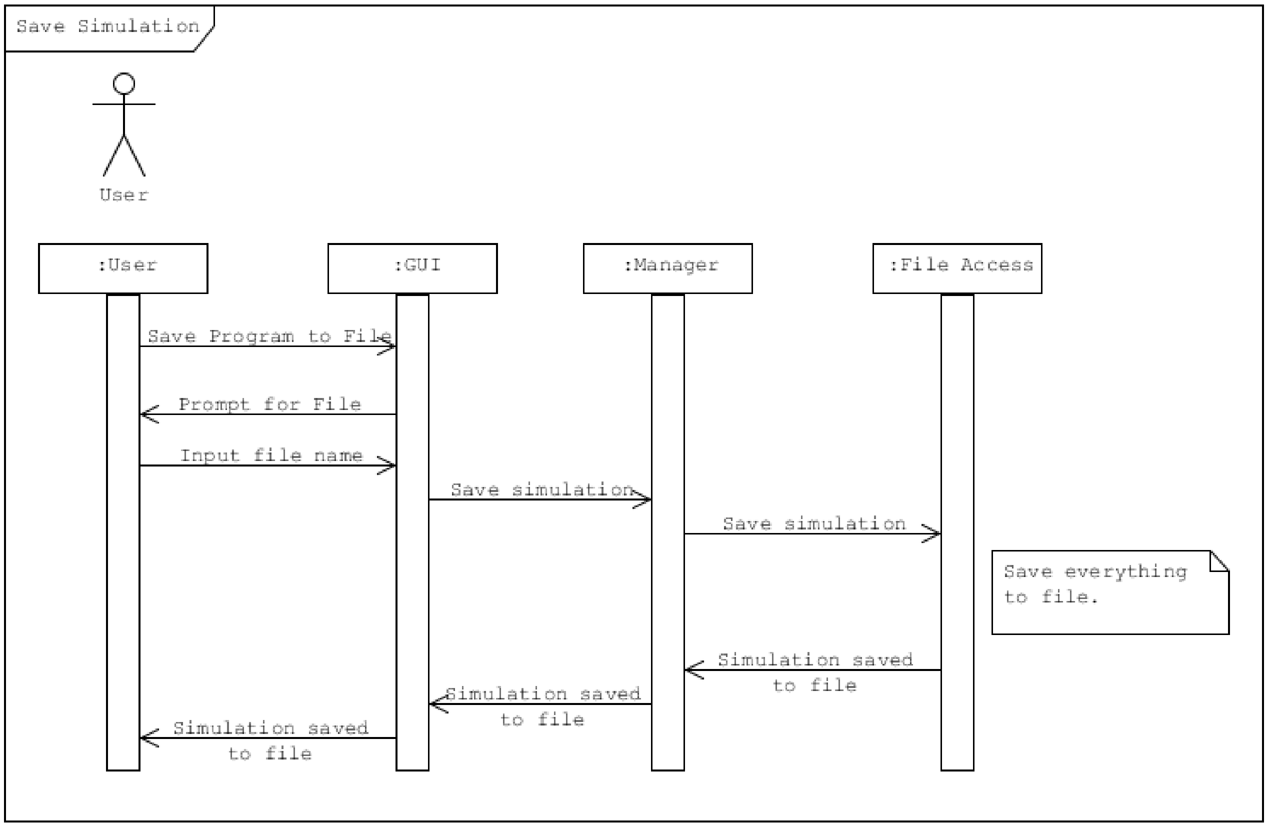


Figure 3. A Sequence Diagram showing the action of saving a program to a file.

5.2.2. Reset Simulation

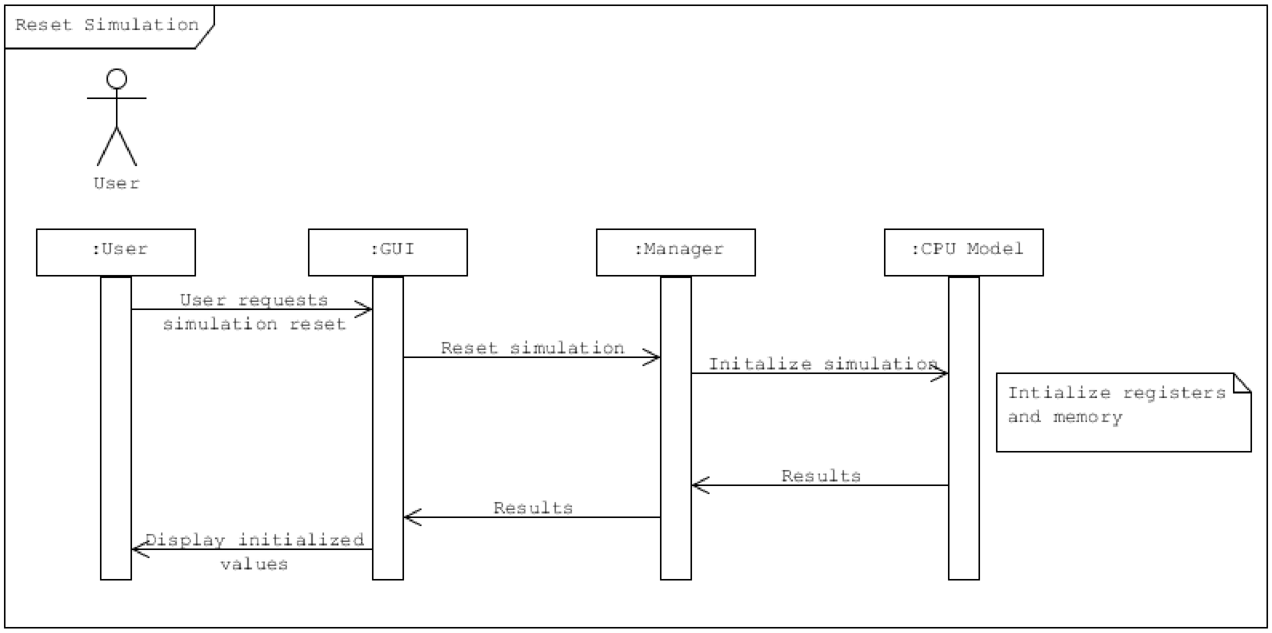


Figure 4. A Sequence Diagram showing the action of resetting the simulation.

5.2.3. Pause Simulation

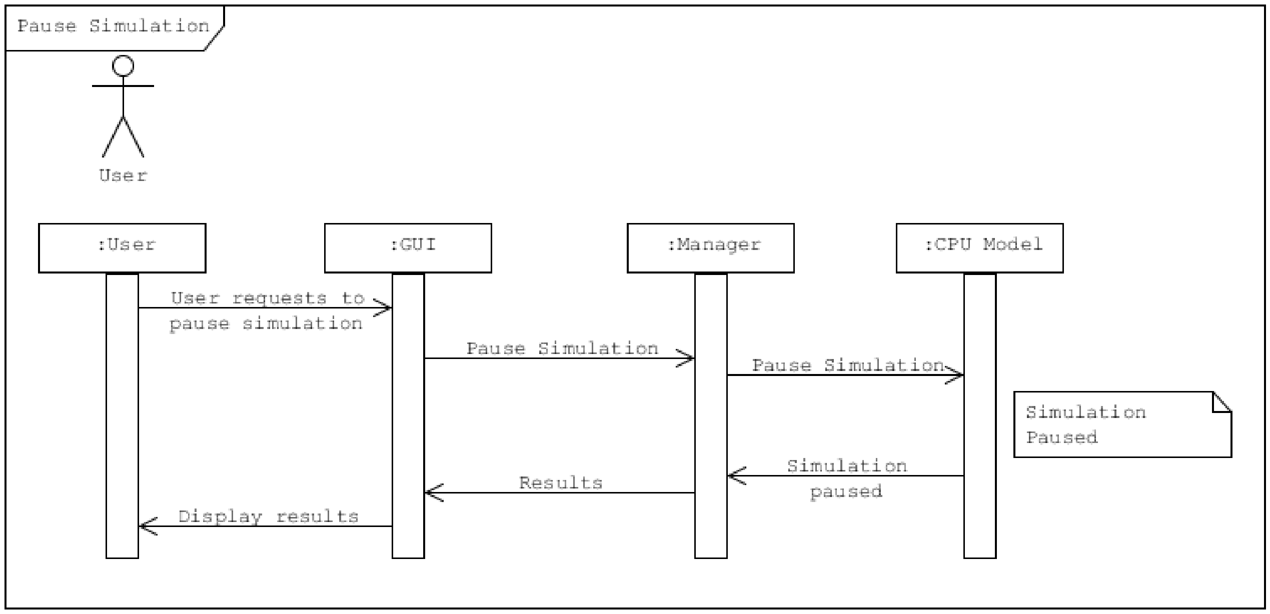


Figure 5. A Sequence Diagram showing the action of pausing the simulation.

5.2.4. Start Simulator

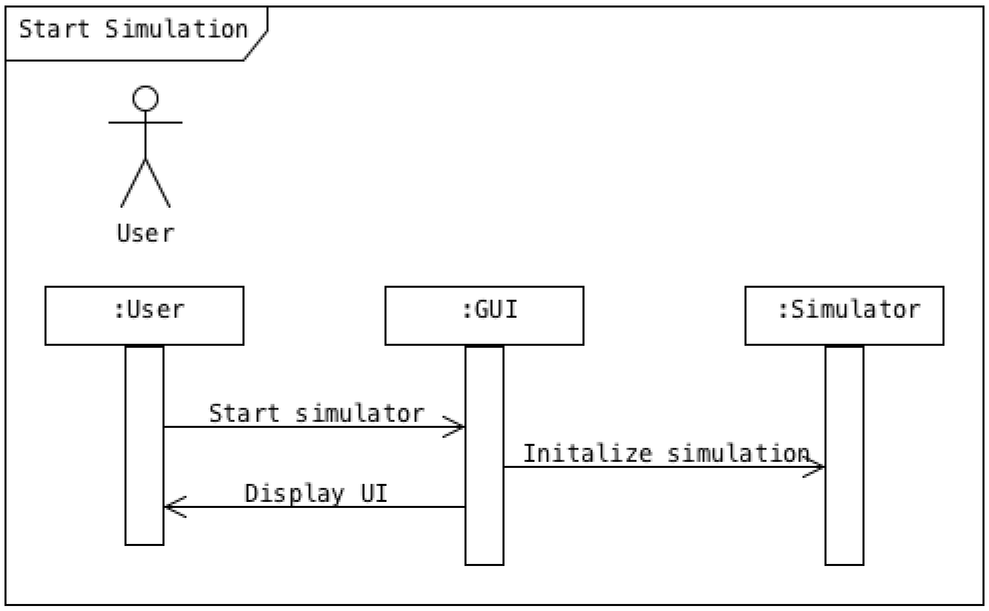


Figure 6. A Sequence Diagram showing the action of starting up the simulator.

5.2.5. Quit Simulator

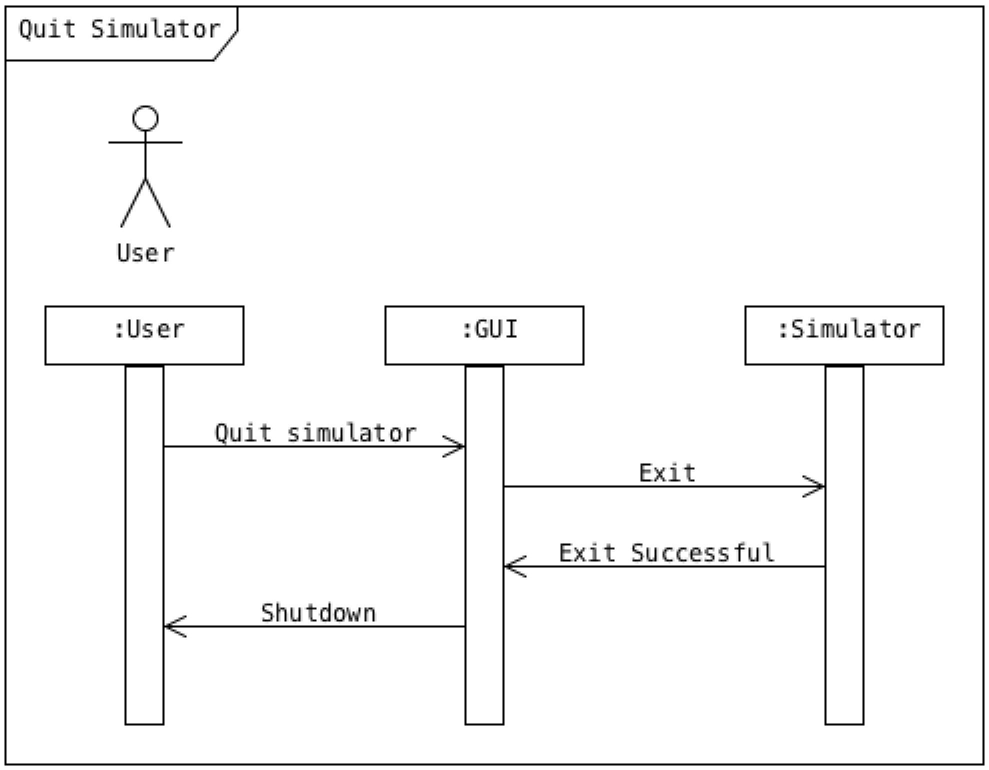


Figure 7. A Sequence Diagram showing the action of quitting the simulator.

5.2.6. Run Simulation

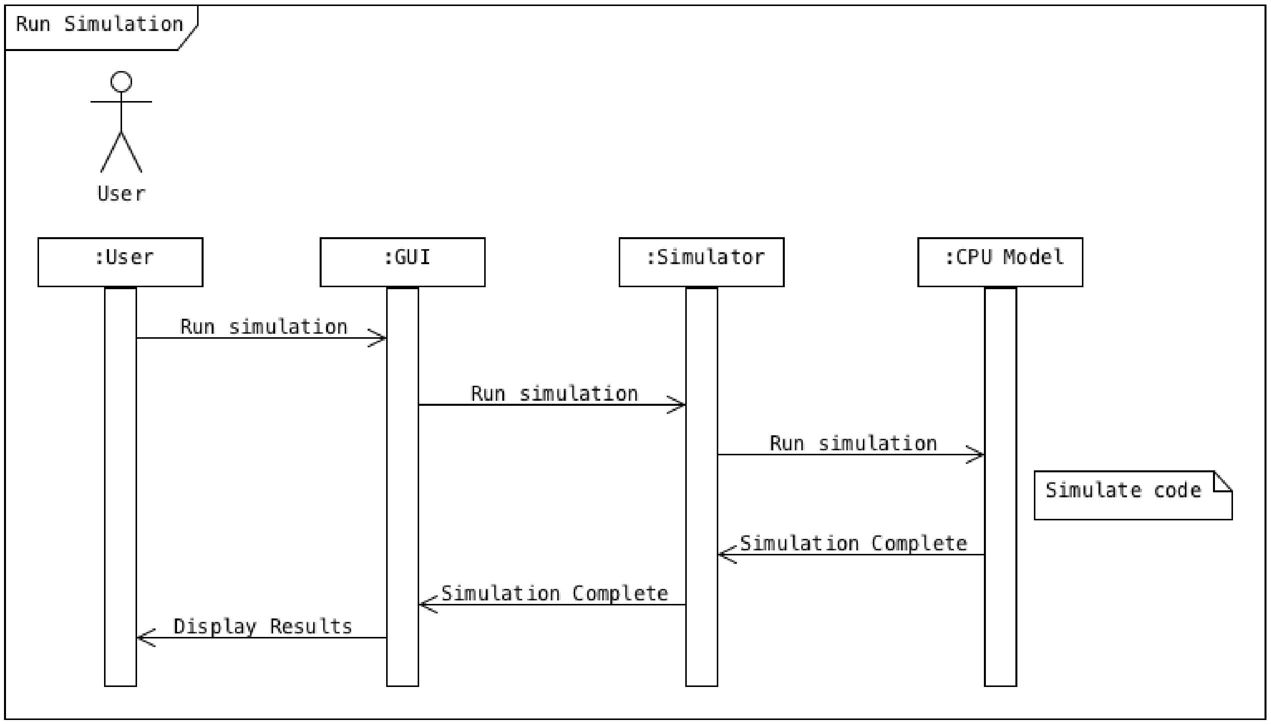


Figure 8. A Sequence Diagram showing the action of running a simulation.

5.2.7. Load Program Into Local Memory

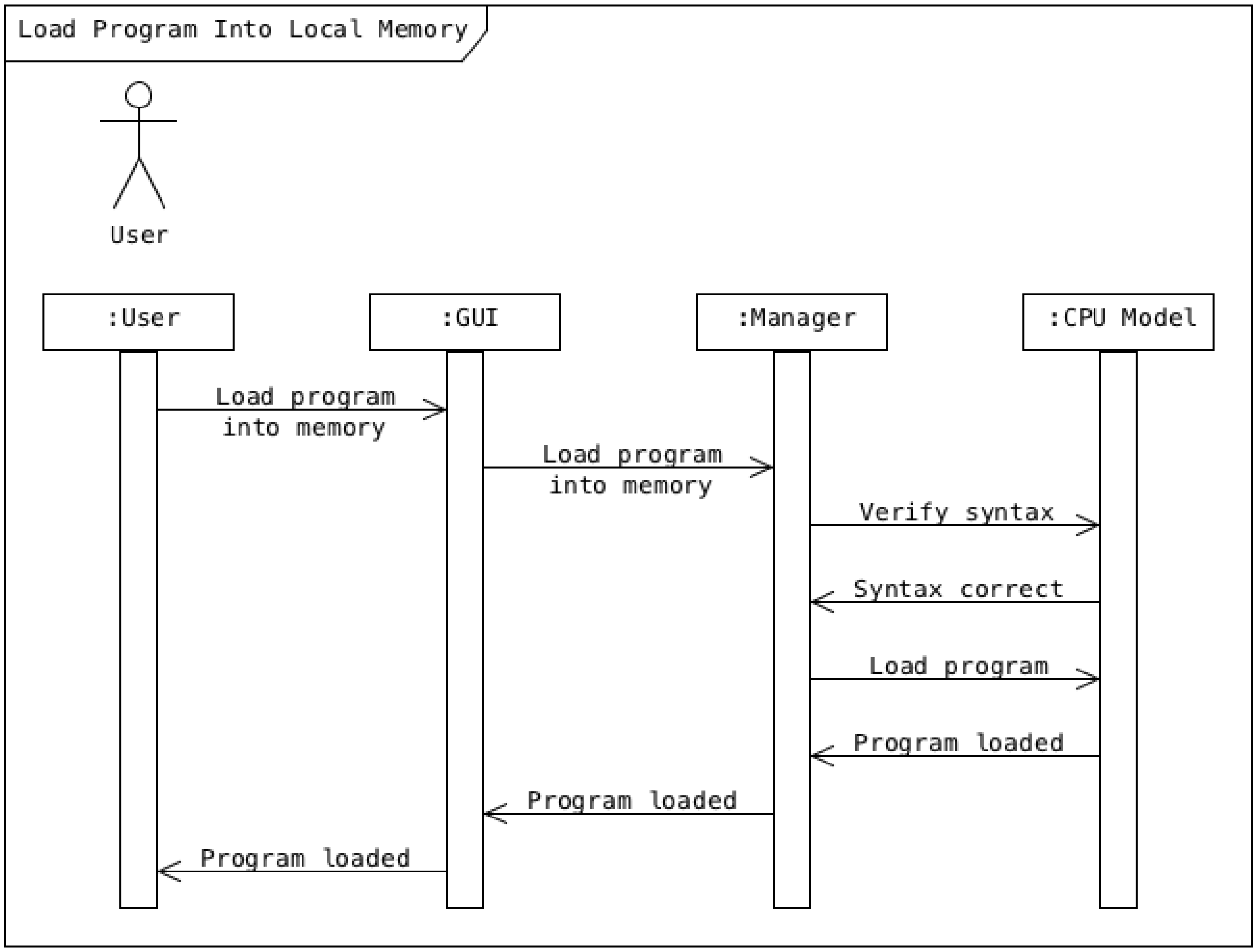


Figure 9. A Sequence Diagram showing the action of loading a program into memory.

5.2.8. Load Program From File

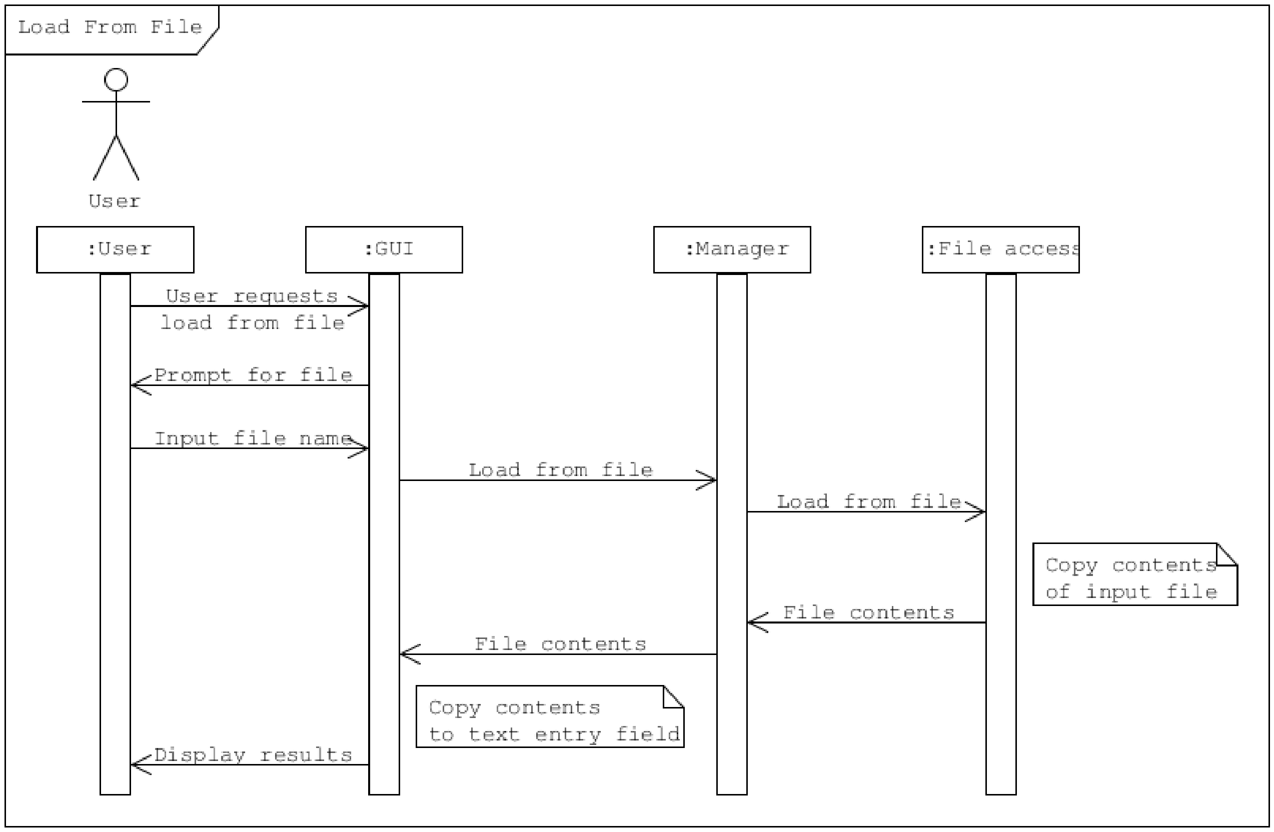


Figure 10. A Sequence Diagram showing the action of loading a program from a saved file.

5.3. Package Diagram

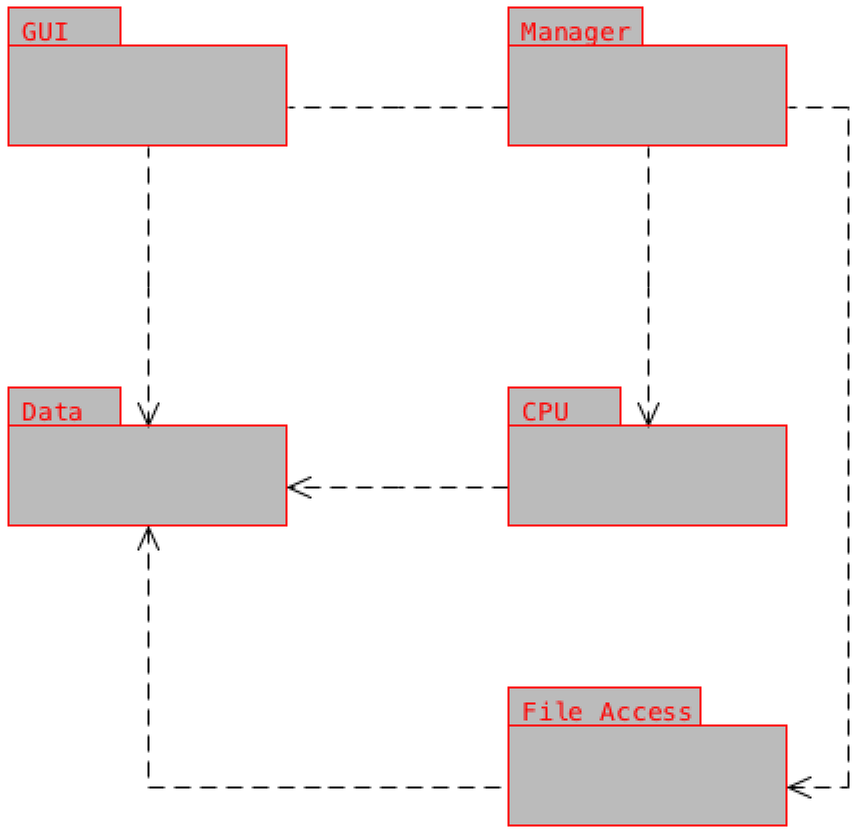
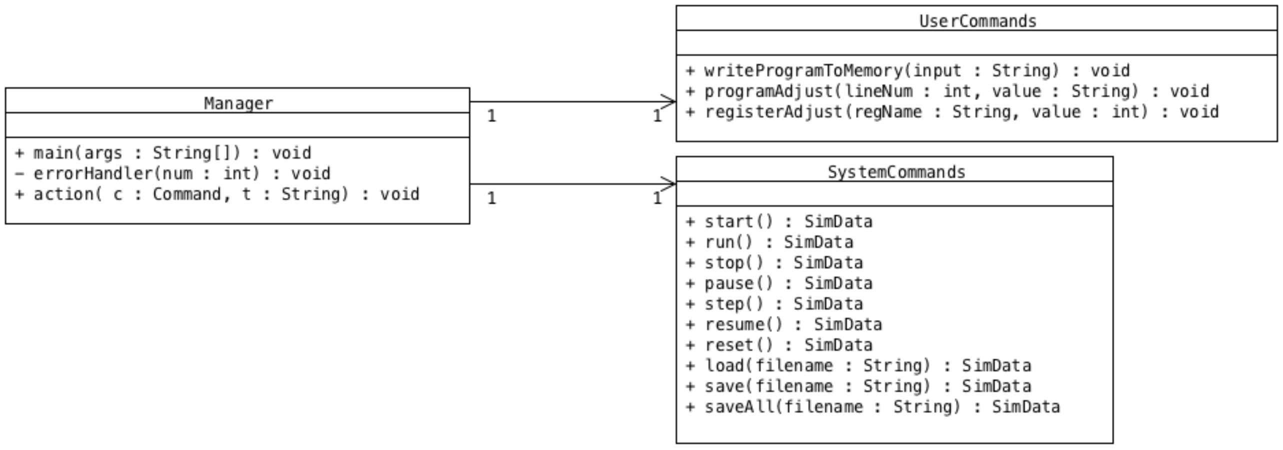


Figure 11. A Package Diagram of the complete system.

This is a package diagram for our system. This is a very high level view of the system and its components. two way communication is represented by a dashed line, one way communication is represented by a dashed line and an arrow. Manager communicates with the GUI, CPU and File Access packages. GUI, CPU, and File Access all communicate with the Data package. The Data package contains all of the information required for the GUI to display, and the information required to save a simulation.

5.4. Class Diagrams

5.4.1. Manager



UserCommands
+ writeProgramToMemory(input : String) : void
+ programAdjust(lineNum : int, value : String) : void
+ registerAdjust(regName : String, value : int) : void
Manager
+ main(args : String[]) : void
- errorHandler(num : int) : void
+ action(c : Command, t : String) : void
1
1
1
1
SystemCommands
+ start() : SimData
+ run() : SimData
+ stop() : SimData
+ pause() : SimData
+ step() : SimData
+ resume() : SimData
+ reset() : SimData
+ load(filename : String) : SimData
+ save(filename : String) : SimData
+ saveAll(filename : String) : SimData

Figure 12. A Class Diagram of Manager.

The manager controls the flow of execution of the simulator. It uses classes UserCommands and SystemCommand to control the simulation itself. the manager has three main methods. Main is where the simulation is created and ran from. errorHandler handles any and all possible errors in the simulation. When the user interacts with the GUI, based on their input actions are performed. Depending on the type of action requested, the manager access UserCommands or SystemCommands to perform these actions.

5.4.2. GUI

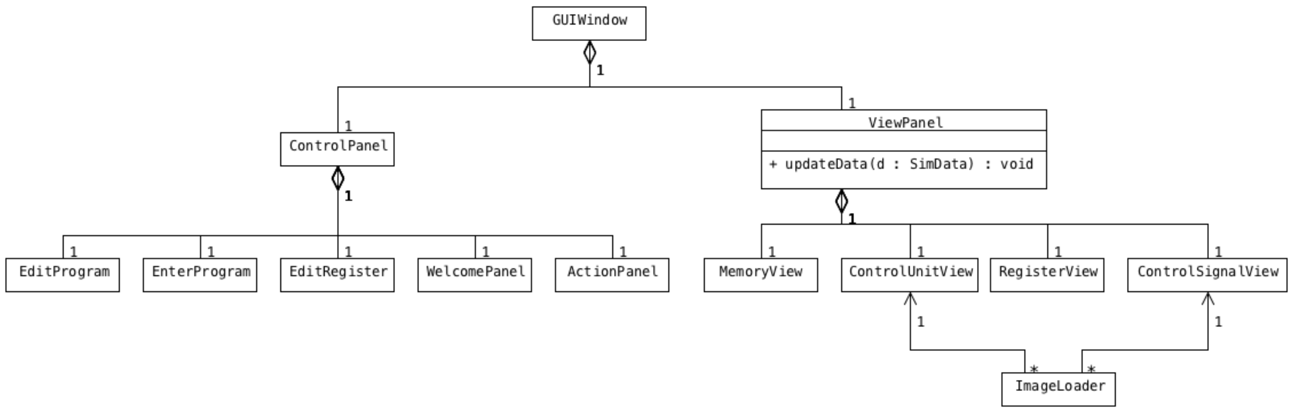


Figure 13. A Class Diagram of the graphical user interface (GUI).

A basic overview of the GUI for the simulator. It shows the different windows and panels within the GUI and how they interact.

5.4.3. Data

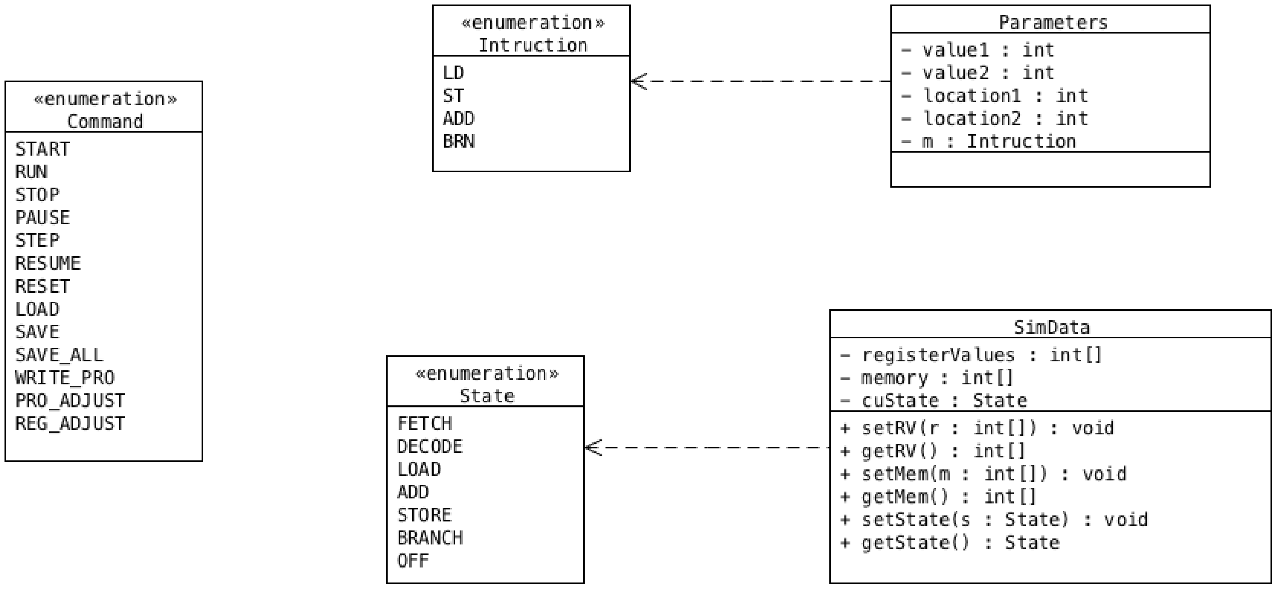


Figure 14. A Class Diagram of the Data objects.

The class diagram for the Data package. The Data package is a collection of all abstract data types(enumerations) used within the simulation as well as the class SimData. SimData is used as the main simulation information storage unit. the GUI will used SimData class to update the GUI appropriately. File Access and Logger will also access SimData in order to save/load a simulation or record noteworthy events.

5.4.4. CPU

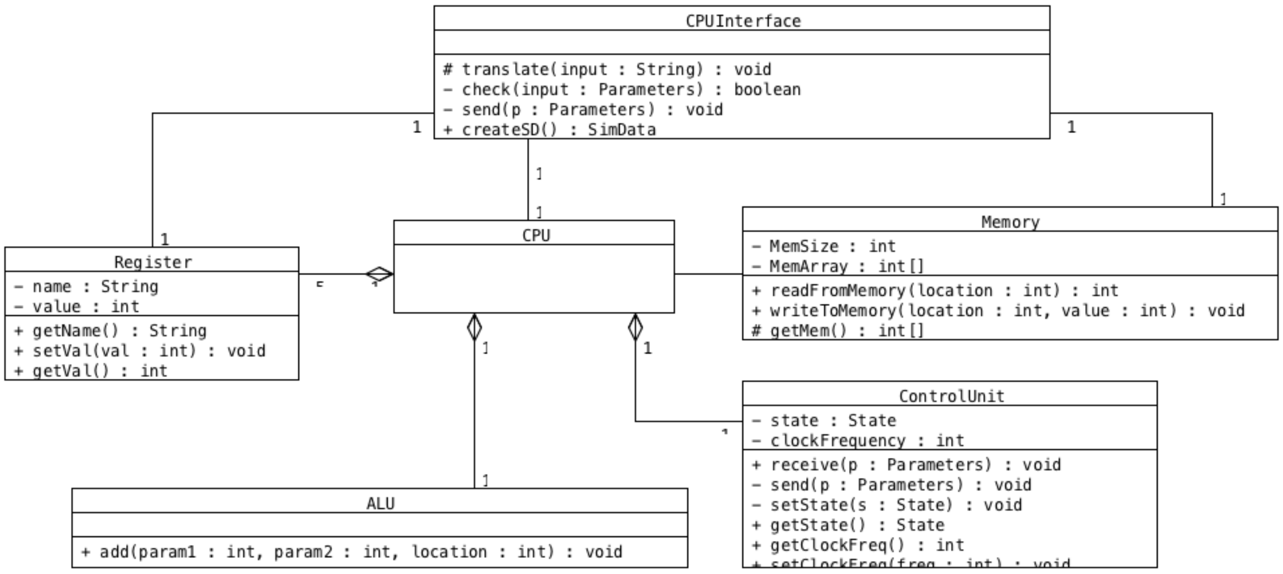

CPUInterface
# translate(input : String) : void
- check(input : Parameters) : boolean
- send(p : Parameters) : void
+ createSD() : SimData
1
1
1
1
1
1
Register
- name : String
- value : int
+ getName() : String
+ setVal(val : int) : void
+ getVal() : int
CPU
Memory
- MemSize : int
- MemArray : int[]
+ readFromMemory(location : int) : int
+ writeToMemory(location : int, value : int) : void
# getMem() : int[]
1
1
1
ControlUnit
- state : State
- clockFrequency : int
+ receive(p : Parameters) : void
- send(p : Parameters) : void
- setState(s : State) : void
+ getState() : State
+ getClockFreq() : int
ALU
+ add(param1 : int, param2 : int, location : int) : void

Figure 15. A Class Diagram of the CPU.

The class Diagram showcasing our CPU Model. The interface ensures that no matter what instructions the manager passes to the CPU Model, the desired output will get returned. the interface also allows for the architecture of the model to change without effecting the rest of the simulation. Past the interface we have the basic components of a computer CPU that will be simulated. The Control Unit, Registers, Arithmetic and Logic Unit, as well as the memory for model.

5.4.5. File Access

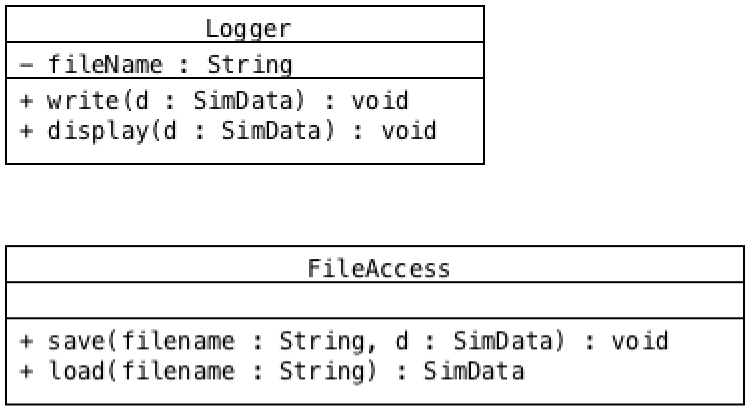

Logger
− fileName : String
+ write(d : SimData) : void
+ display(d : SimData) : void
FileAccess
+ save(filename : String, d : SimData) : void
+ load(filename : String) : SimData

Figure 16. A Class Diagram of File Access.

The File Access package class diagram is simple. Within the package there are two classes, the Logger, and FileAccess. Logger writes noteworthy events to an external text file that can later be used for marking, or debugging. File Access accesses external files by either loading said files, or saving a current simulation to an external file that can be accessed at a later time.

5.5. State Diagram

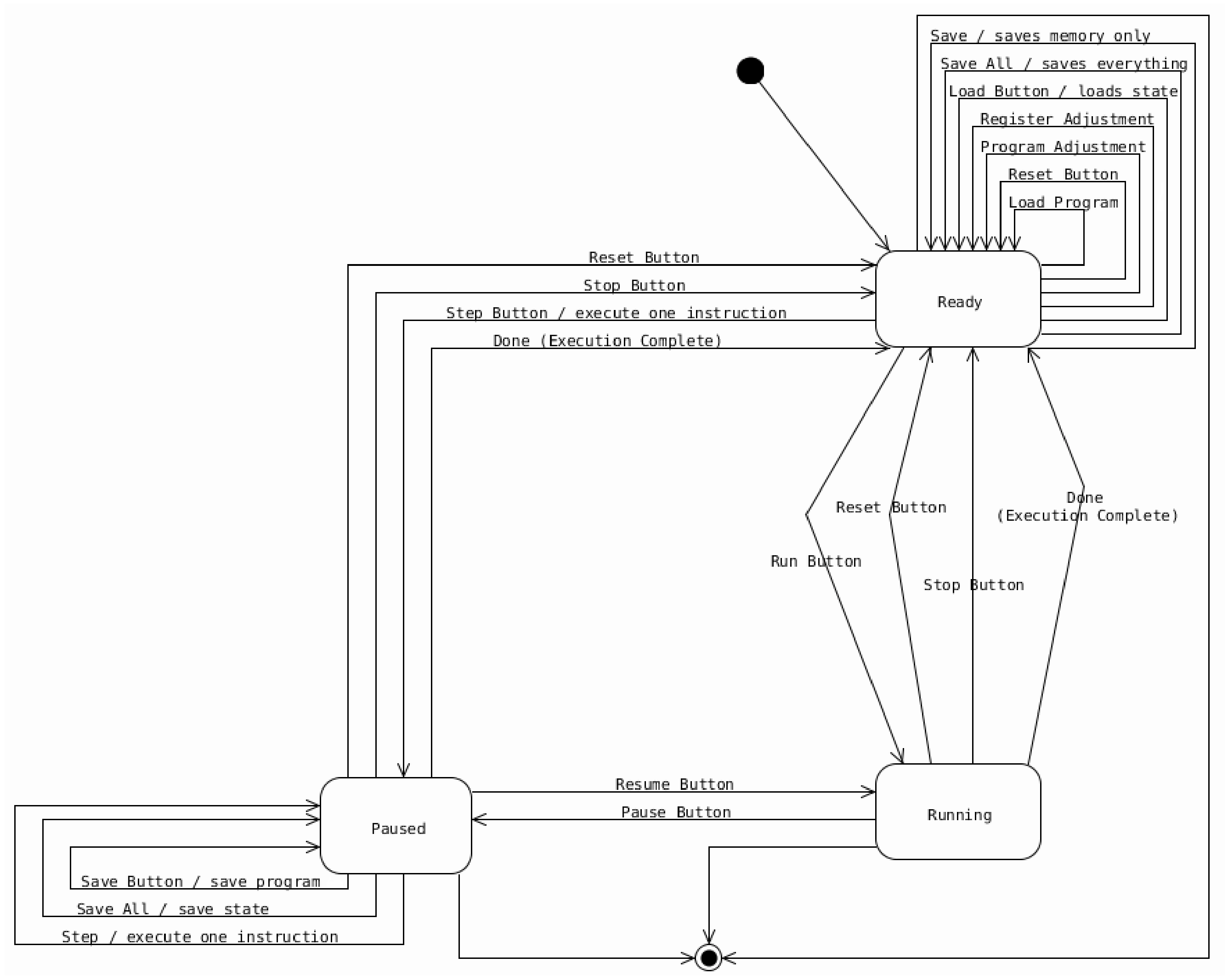
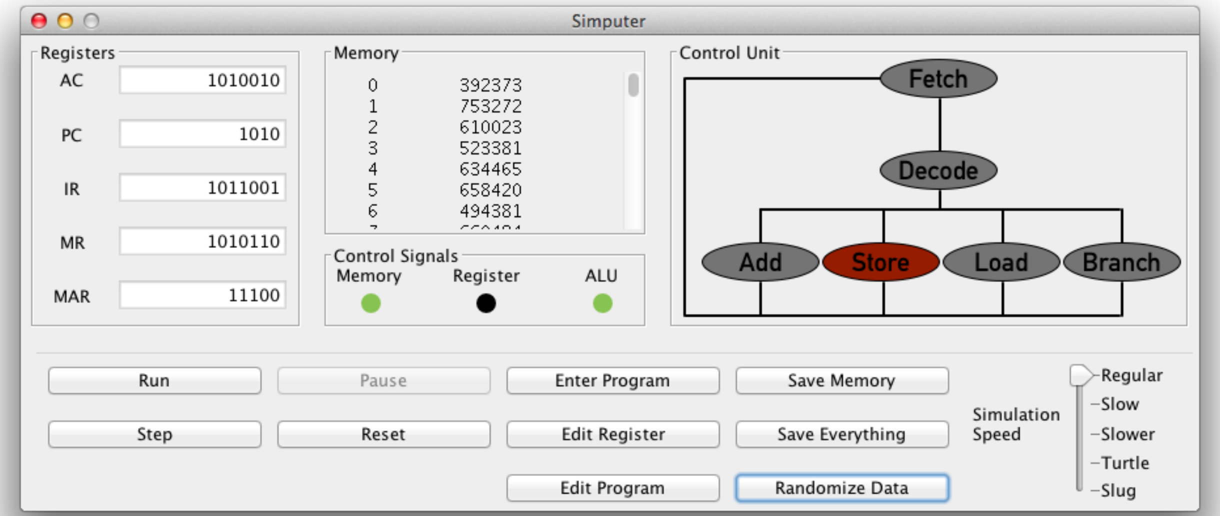


Figure 17. A State Diagram of the system.

The state diagram for the simulator. There are three states, Ready(1.2.8.1), Running(1.2.8.2) and Paused(1.2.8.3). The simulation transitions through the states based on the events above.

6. System Prototype



AC	1010010
PC	1010
IR	1011001
MR	1010110
MAR	11100

```
0    392373
1    753272
2    610023
3    523381
4    634465
5    658420
6    494381
```

Figure 18. A prototype of the GUI that the user will interface with.

In Figure 16, the prototype of the GUI is displayed. The user has complete control of the simulation from this window. The Registers box displays the value store in each of the five registers in binary numbers. The Memory box displays all the values stored in the memory, this panel is scrollable. The Control Signals box displays what signals are being sent to the ALU. The Control Unit box displays the current state at which the Control Unit is in. The lower third of the window are user controls. All controls are defined in 2.2.1 and 2.2.2.