|  |
| --- |
| CSCI 6461 Semester Project |
| CISC Computer Simulator – Design Notes |
| Manuel Pérez González, Mashaal Al Mazro’ei, Pedro Rios Alvarez |

Overview of project: The object of this project is the implementation of a simple CISC-based computer simulator. This simulator is not the implementation of any real computer architecture or its instruction but will be implemented and used as a tool to illustrate how instructions are processed and stored.

(*From the Project Description*)  
It has the following characteristics – for Phase I:

* 4 General Purpose Registers (GPRs) – each 16 bits in length
* 3 Index Registers – 12 bits in length
* 16-bit words
* Memory of 2048 words, expandable to 4096 words
* Word addressable

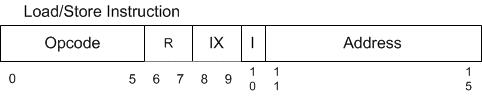
# Technology and tools used

1. IDE: Eclipse (Mars)
2. Language: Java
   1. Graphic Libraries: Swing (Builder Tool)
   2. API: None\*\*
3. Repository and CVS: Git and GitHub.com

**Notes:** \*\*The team decided to develop our own classes to implement the instructions due to time constrains. Basically it came down to the decision to devote time to understand the material and the scope of the project versus trying to learn and implement a new API.

# Design considerations

## Part I

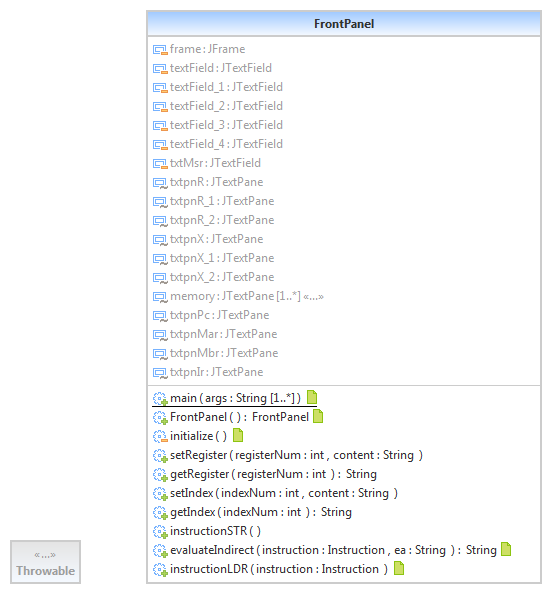
1. **Main consideration:** The design for this project and this part in particular was determined in part by our present understanding of the Von Neumann Architecture and how instructions are processed. The team tried to focus on ease of implementation on the instructions on the simulator as opposed to being strict on aspects such as bit-wise operations or internal memory assignment process based purely on numbers.
2. Input: The input for the simulation reflects the Input/Output structure described in the Project Description document:  
   

Each of the input fields corresponds to the fields of the instruction. The other option was to take the instruction as a whole and then parse the input internally but that would add more processing code that we could avoid if we just separate the different fields at the UI level. Other considerations include:

1. Make easier for the user to input information for the different instructions using the familiar format of the instruction.
2. A tool for training: Help the new user to become familiar with the general format of an instruction and how each input affects the result or output of the instructions that are being processed.
3. Separating the input was really the only feasible way to process each instruction by combining and processing each piece of input in order to determine precisely the type of operation required.
4. Testing and error handling**:** At this early point testing requirement are basic and limited to the essential functions to implement each instruction. Error handling is handled internally by the use of the try and catch techniques in the code. Output error messaging and condition codes (CC) are still not being implemented at this point.
5. Data types and data structures**:** In order to facilitate the implementation of the instructions all inputs and internal data handling will be processed as Strings. The string inputs will be returned to as binary numbers for conversion purposes. In future versions the user should be able to view the information in the output fields in numbers from different bases. It is possible that the process of using Strings as data type could change as output requirements and instruction processing will increase in complexity and thus would require the use of a numerical data type.

The project centers on the UI file (‘FrontPanel.java’). This file acts as the ‘main’ method and it connects the classes that implement the instruction.

1. Class overview**:**
   * FrontPanel **– Serve as the main class for the UI and also run and holds all the helper classes that run the simulator.**
   * Instruction **– Simulates the instruction on the machine. Its variables hold the reference to the logic of each instruction.**
   * InstructionEnum **– Holds the relationships of the OpCodes. This class refers to the OpCodes by substituting their numeric code with their string equivalent (i.e. ‘LDR’, ‘STR, etc.). This is mainly to facilitate the understanding of the code.**
   * BinaryUtil **– Provides generic functions to manage of binary numbers in the application.**

****

Part 2

Main consideration

The design of this part of the project centered mainly in implementing the instructions and UI elements to support the execution and running of Program 1. Following is a description of Program 1 as per the Project Description provided by Prof. Lancaster:  
 *Program 1: A program that reads 20 numbers (integers) from the keyboard, prints the numbers to the console printer, requests a number from the user, and searches the 20 numbers read in for the number closest to the number entered by the user. Print the number entered by the user and the number closest to that number. Your numbers should not be 1…10, but distributed over the range of 1 … 65,536. Therefore, as you read a character in, you need to check it is a digit, convert it to a number, and assemble the integer.*

Input

In addition to the input featured in the first part the UI now can accept input directly into memory for testing purposes as well as in any of the other input fields like those on the registers and the indexes. Now for a more convenient way to test the different instructions we have created a drop down menu that will load pre-selected values into the different areas that a specific instruction will affect.

## Testing and error handling

The testing for the instruction was done by direct and manual input using the interface. This was designed to give immediate feedback by using the ‘Single instruction’ feature and adjusting the behavior of the instruction for the display as well as internally as well.

For error handling we created some provisions to handle some issues such as overflow and underflow but the formal mechanisms to handle and testing errors will be implemented and added on Part 3 with the addition of the trap instructions. We are also expecting to add a more robust way to handle errors and to receive feedback in the interface.

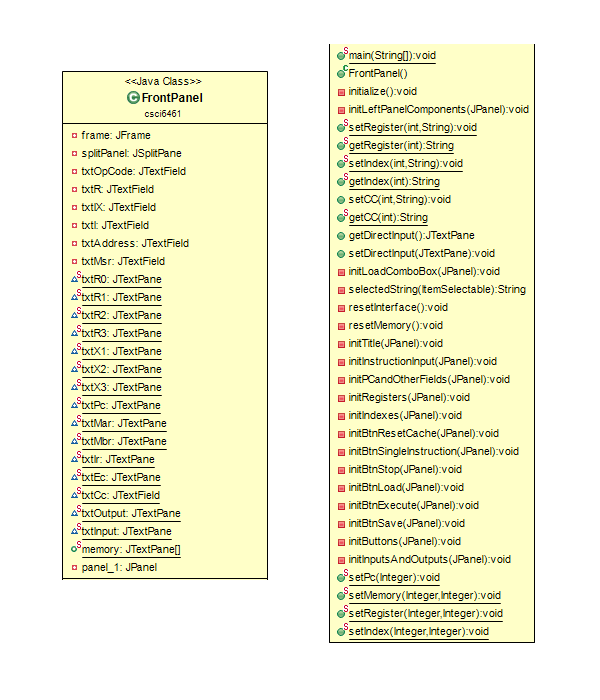
## Data types and data structures

In this part we have used Java’s classes to help process the instructions and also we have created some methods to handle some of the tasks behind the scenes. These methods are considered helper methods and they perform mostly conversion tasks to handle the strings and integers and to convert them into binaries for processing and display in the UI. We will go briefly over them when we describe each class.

## Class overview

### Main classes: UI

#### FrontPanel



The Front Panel class connect most of the functions of the simulator under the user interface (UI). Due to the expanding requirements for this part this class has grown in complexity and size despite our best efforts to keep components separate for ease of maintenance and understanding.

Among the new features for this part are:

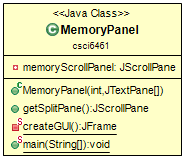
* New memory panel – This new component reflects the increased use of memory by instructions and also the implementation of the cache memory.
* Instruction and program menu – This new feature was created to help test the instructions directly on the UI as well as to facilitate the use and execution of Program 1 by the user and testers.

#### HardCodeBuilder

#### 

The HardCodeBuilder is the class behind the drop down menu for the different instructions implemented as well as for Program 1. This class was created to facilitate the testing of the instruction in the UI and their corresponding values in the areas of the panel such as indexes and registers. This also includes specific instructions and their corresponding address values.

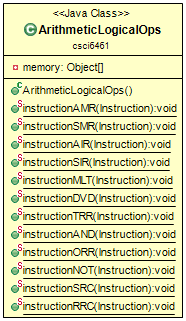
#### MemoryPanel



The MemoryPanel class manages the display and functions of the main and cache memory in the UI. It consist of a list of the memory positions list that the user can scroll.

### Main classes: Instructions

#### ArithmeticLogicOps



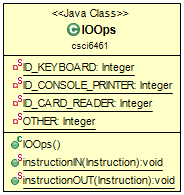
The ArithmeticLogicOps class is the implementation of the Arithmetic Logic Ops instructions.

#### FloatingPointVectorOps

#### 

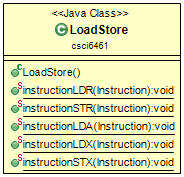
The FloatingPointVectorOps class implements the instructions for the Floating Point and Vector Operation that will be implemented in Part IV.

IOOps



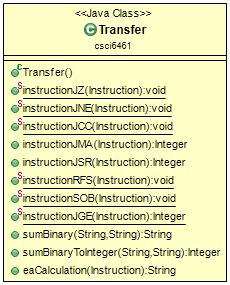
The IOOps class implements the instructions for the Input and Output instructions.

#### LoadStore



The LoadStore class implements the Load and Store Operations instructions.

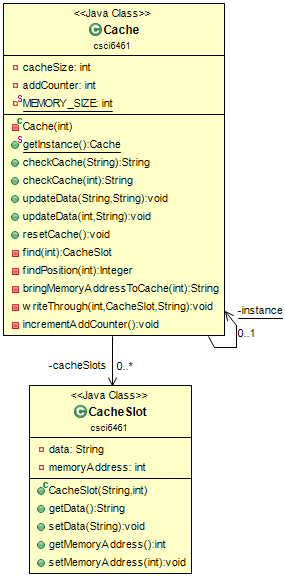
#### Transfer



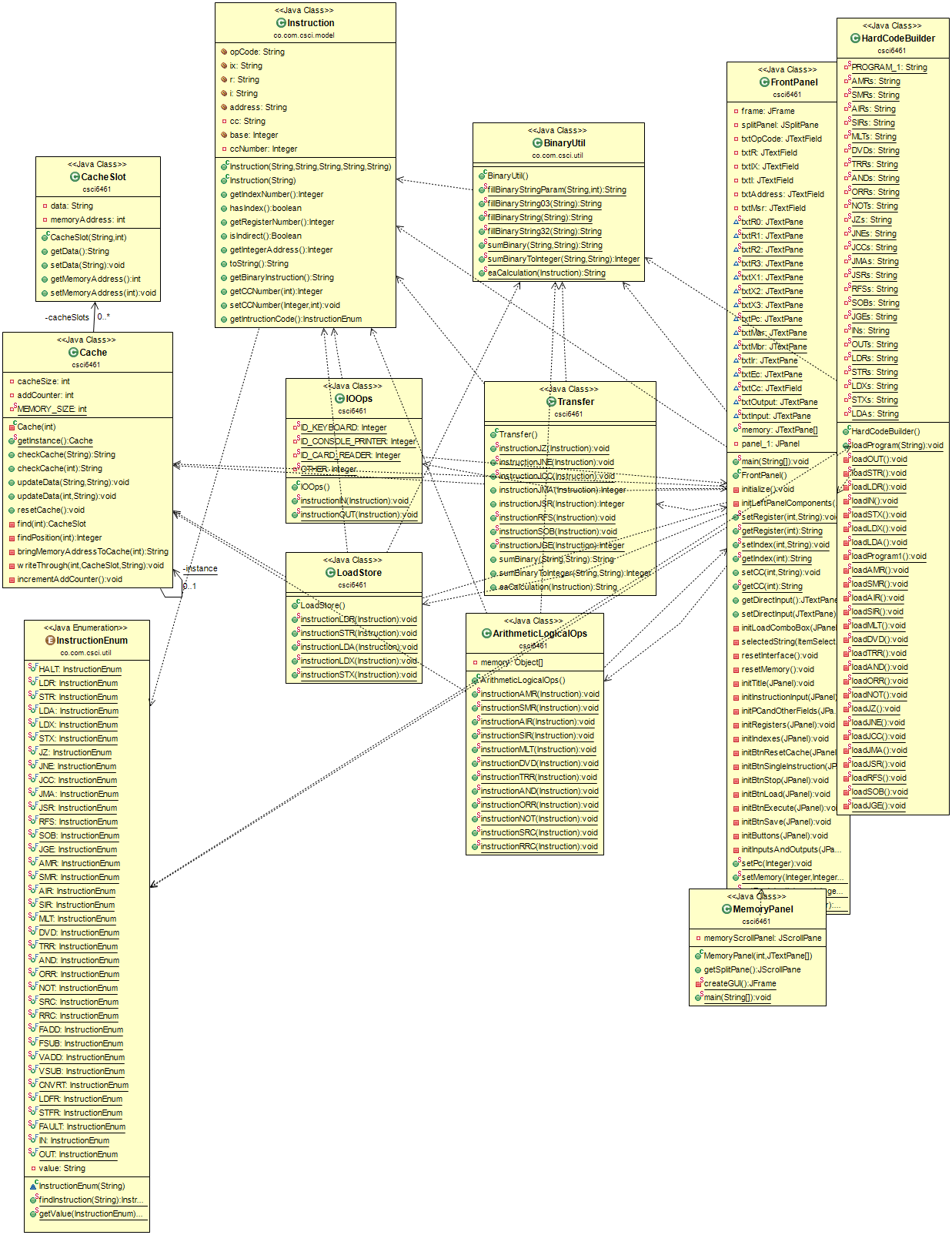
The Transfer class implements the Transfer Operations instructions.

### Main classes: Auxiliaries

#### Cache



The Cache class implements the cache memory for the project. It has functions that will check and update the contents of the cache. It uses the write-through method to handle content that is written into the cache. This method updates main memory of the content that is written into memory in order to maintain consistency between the two.



Part 3

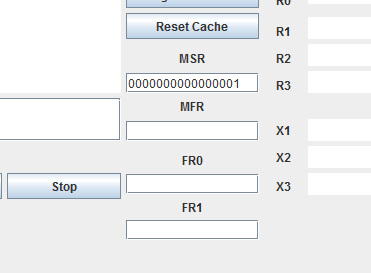
Main consideration

For this part the project didn’t change heavily from a design point of view. The class FileReaderUtil was added to be used from the HardCodeBuilder to allow the program #2 read the paragraph from a file called “TextProgramTwo.txt”. This new class only has one method and no variables. The method getFile which receive the name of the file. Also the class TrapCode was added where it’s managed the TrapCode logic of the simulator following the logic of the previous instruction classes.

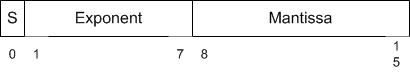
Part 4

Main consideration

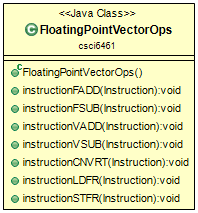
The implementation of Floating Point and Vector operations was the main objective of this part. Instructions were integrated into the existing user interface and thus the floating point registers FR0 and FR1 were added in order to store float point values while carrying out the instructions for floating point operations.



These instructions are contained in the class follow the format for all parameters on the methods that implement the individual instructions:



The ‘FloatingPointVectorOps’ class implements the floating and vector operations.

****

The instructions implemented are:

FADD – Floating Point Addition

FSUB – Floating Point Subtraction

VADD – Vector Addition

VSUB – Vector Subtraction

CNVRT – Convert to Fixed Floating Point

LDFR – Load FP Register

STFR – Store FP Register

The class ‘FloatRepresentation’ handles the operations in the backend on the program. It perform the calculations for the exponent and mantissa components for the instructions in order for the program to process them into their respective memory locations.  
  
