|  |
| --- |
| 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

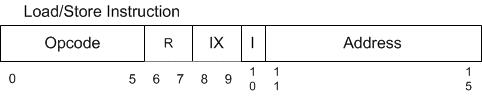
# Technology and tools used

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

**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

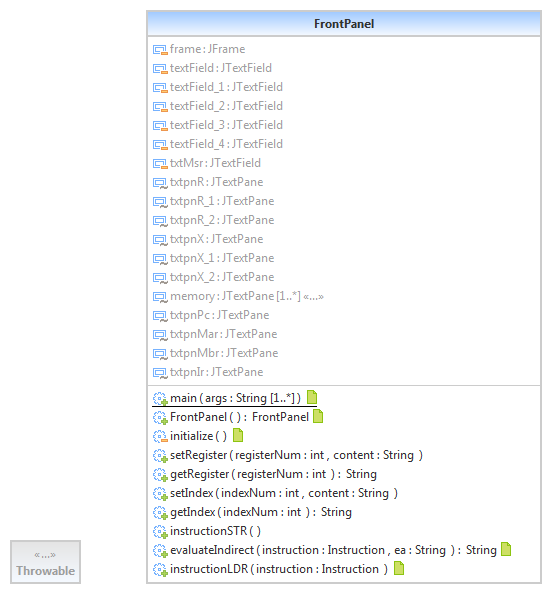
1. **Main consideration:** The design for this project and this part in particular was determined in part by our present understanding of the Von Neumann Architecture and how instructions are processed. The team tried to focus on ease of implementation on the instructions on the simulator as opposed 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.**

****