CSCI6461 – Computer System Architecture
Computer Simulator – Component 1

The George Washington University
Spring 2017

Group Members (ordered alphabetically):

Pedram Hosseini
Zhihong Lian
Yanbo Li
Ziyi Wang

NOTE:

- We used GitHub as a tool for controlling and managing our source code. You can access all codes of our project using the following link: https://github.com/phosseini/CSCI6461-computer-architecture
- I have also attached a video in our package. I think it would be of great help if you watch the video to be familiar with different parts of our code and how you can run and use our simulator.

How to use the simulator:

After running the project, you will see a user interface. You have the following options in the interface:

1. You can store any value in any register that you want. For doing that, you should select/deselect the radio buttons that you see in front of each register. Please note that you cannot enter a value in the textbox in front of the register and you should select the value only using radio buttons. After that you selected the value, you simply can push "**store**" button to save the selected value into the register. **Note:** a selected radio button is equal to 1, and a deselected radio button is equal to 0.
2. We have **a memory interface**. Using this memory interface, you will be able to either store or load a value to/from memory. In order to store a value into memory, you should enter a **valid address** (an address between 0 and 2047) into address textbox and a value between 0 and 65535 into the value textbox. If you want to load the content of a specific address of memory into value textbox, you can simply enter a valid address into the address textbox and push the load button and you will see the content of that memory address in the value textbox.
3. There is a button named **IPL** (initial program load) in our user interface. By pushing IPL, the machine pre loads a program that shows that the machine works and stops at the beginning, ready for the user to hit execute. When you push the IPL button, you will see some values for some registers and also some messages in our console. These messages that you see in console are the instructions that we executed to show that our machine works fine and the values that you see for different registers are the values after executing the mentioned instructions. It is worth pointing out that we use the **console** in our user interface to show some custom messages. For example, when you store 5 into the R0, the first general purpose register, you will see a message like "**R0 is set to: 5**" in console.
4. The main part of the simulator is executing the instruction. There are 16 radio buttons in front of instruction that correspond to 16 bits of an instruction. The same as selecting/deselecting the registers, you can enter an instruction. Afterwards, you can push "**execute**" to execute the instruction. After pushing the execute button, the value of all registers will be updated accordingly. For example, if you have load instruction that load

something into one of the index registers, after executing that instruction, the value related to that index register will be shown on user interface in front of that specific index registers. The value of any other register that has been affected by executing that instruction will also be updated and shown.

Different parts of code and design note:

It is worth pointing out that we tried to use proper comments entire our code to make it easier for any reviewer to understand our code. We have 5 main packages in our code that I briefly explain what the intention was behind creating each of them.

1. **ALU** (arithmetic logic unit): In this package, we have two classes. One of the classes named **Instruction.java** is created to be used as an object for the instruction that we are going to execute. When we enter an instruction to execute it, sometimes we need to pass the instruction to different classes and different methods. As a result, we created this class and we store our instruction with 5 fields namely including address, I, ix, r, and address and use this object as an argument when we need it in our code across different methods and classes.
   Another class in this package is the **Instructions.java** class. This class is the main part in our ALU package. We have three methods in this class. The first one is called **bin2dec** that simply converts a binary value to a decimal value. The second method is the **execute** method which executes an instruction. We pass an instruction, an object of registers class, and an object of memory control unit class to this method to execute the instruction. And, the third method in this class is named **EA** that we use it to calculate the effective address.
2. **Front**: To cut a long story short, this package has one class named **FrontPanel.java** and we manage everything that is related to the user interface in this class.
3. **Memory**: in this package, we have a class named **MCU.java** that stands for Memory Control Unit. Everything that is related to memory will be managed in this class. In particular, we have three main methods in this class for fetching a word from memory, storing a word into memory, and load from read only memory (ROM). This last method is used to load the instruction and the address in which we are going to store the instruction and we call it when the user push the IPL button. We also have two other methods for reading the current size of our memory and expanding the size of memory. We mainly designed these two methods for future components that we should expand the size of our machine memory.
4. **Registers**: The same as Memory, we have one class in this package named **registers.java** and we manage everything that is related to all the registers in this class.
5. **Util**: This package contains a class named **Const.java** and is designed for keeping some configurations related to opcode.