The purpose of this project is to develop a program that simulates a simple computer with CPU and Memory. There are total 2 processes: 1 for CPU and 1 for Memory. To achieve that on Java, it’s possible to use Runtime#exec() method. Memory part handles only simple read/write operations with integers. CPU includes registers and it’s able to process multiple instructions.

First of all, I defined one interface called Instruction, since both of Memory and CPU components are based on the different instructions and have a common behavior (read/write methods). So that Memory class implements this interface and provides its own implementation of these methods. At the same time CPU class implements this interface as well and provides implementation of methods. This is the main logic of my design.

To write data into the memory, WRITE\_COMMAND is used along with the address and information (all commands and instructions are placed in the Instruction interface). To read data from the memory, READ\_COMMAND is used with address from where information should be read.

Class Memory encapsulates a memory part of this simulation. It contains array as an instance variable that stores up to 2000 integer entries. Since the data to be stored can only be integers, it doesn’t need to create more complicated data structure to represent a memory. This class has a constructor to instantiate a new Memory object and to initialize it with the file name to be read. Input file is read through the Scanner line by line.

Class CPU includes multiple instance variables, such that PC, SP, IR, AC, X, Y (standard registers), timer, timeout (to handle interruption and switch kernel mode), kernel mode Boolean variable (true if it’s possible to execute any CPU instruction), and two objects that are responsible for the read/write operations.

The main method in CPU class is process (), which processes registers according to the instructions. It should be noted, that Memory has its own process, so CPU and Memory work separately (it’s important). Every time when next instruction is read, registers get a new value. For example, if IR (or program counter) meets value = 1 (or LOAD\_VALUE), then IR should be updated by reading the next value from memory and accumulating the new value (AC). In other words, memory is organized as a stack where values can be pushed / popped. When kernel mode is true, it’s possible to execute any CPU instruction, otherwise it needs to return from system call I learned a lot about internal processes in the computer; realized a simple model that allows simulating the base communications between CPU and memory.