**Project 1**

The project have three packages including components, conversion and gui. Components consists of all kinds of components of CPU and the Memory. The conversion consists of three static function that can convert binary, decimal, and hexadecimal.

In components, the class CPU\_control controls all the components' objects, can decode the instruction, and execute those instructions. The class interact can be used by the GUI which consists of an object CPU\_control that can set data, get data, execute instructions, and return status of potential errors.

**Classes of components:**

Specifically, the in class General\_Purpose\_Registers, the class attributes include an

private array which include 4 integers. The function getregister can get the value of the registers and can checks if the index value is between 0 and 3 and then grabs it. The setregister function sets the register according to the new value being passed in and limit the index and the value.

In class Index\_Registers, the class attributes include a private array which include 3 integers. The function getregister fetches the index address if it is between 1 and 3. The function setregister can set the IXRs from GUI and limit the index and value.

In class Instruction\_Register, the class attributes include integers of instruction and all parts of the general instruction. The function setinstruction sets all the appropriate values according to the instruction as a part of decoding process. Additionally, the function can return each part of the instruction as required.

In class Machine\_Fault\_Register, the class attributes include an integer of Faultindex which can be set to 0, 2, and 3. The function getFault can return the Faultindex. The setFault can set the Faultindex if appropriate given the context. Finally, the resetMFR can reset the Faultindex to -1.

In class Memory\_Address\_Register, the class attributes include the integer of the address of Memory that the CPU needs to access. The function getMemaddress gets the returns the current address in the MAR. The function setMemaddress Gets the new address, determines if it is appropriate, then sets.

In class Memory\_Buffer\_Register, the class attribues include an integer of Data. The function getData gets the returns the current data in the MBR. The function setData Gets the new Data, determines if it is appropriate, then sets.

In class ProgramCounter, the class attributes include an integer of PCaddress. The function ProgramCounter can reset the PC if appropriate. The function PCPlus increments the PC and the function getPCaddress gets the PC address. The function setPCaddress re-assigns the PC address if required and fits the appropriate range of the machine.

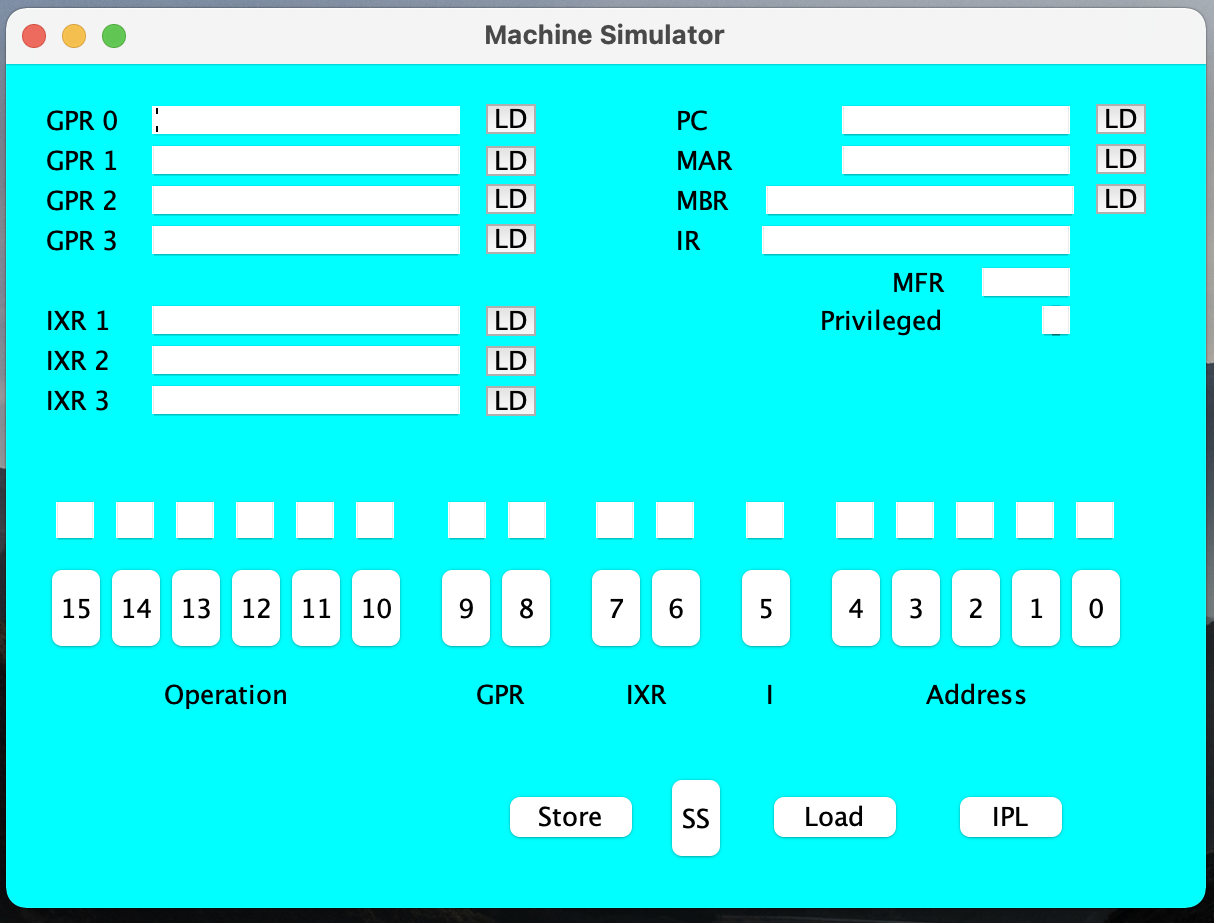
In class Memory, the class attributes include an array of integers to store the memory data and an integer start which indiciates the start location after the reserved locations. The function Memory can reset the memory if the size needs to be bigger than 2048. The function readMem can return the data of memory, it can determines if the memory address is acceptable or not, and finally can throw errors in applicable. The function writeMem can write to memory if the address is within the acceptable range and if the new data is within the bit-limits of the project. The function CPUaccess and CPUwrite are used by CPU\_control to access the reserved locations because the CPU needs to handle the machine fault.

In class CPU\_Control, the class attributes include all the components classes above and an integer mfindex which is used to determine the machine fault and an integer halt which can be sent to the GUI and tell the GUI to halt and block the runsinglestep function and runinstruction function. The function initial sets the initial components of the machine which can start or restart the machine and reset halt and read IPX.txt and load it to the memory. The function runsinglestep can run the instruction where the PC points one by one while also fetching, decoding and executing. The function runinstruction can run the current instruction in IR without fetching and it is used to execute the load and store button. The function load acts as the load instruction. The function store acts as the store instruction as well as the function LDA, LDX, STX. The function checkaddress can check the access of memory is written or not. If not, it go to set the MFR and get the solution (which is halt right now) and will be called after each read and write to the memory. The function machinefault deals with the machine fault and it will find the solution's address (6 for hlat) and get the solution instruction (again – ***6)***.

The class interact is the class the GUI can use and all the components are in the objects in the CPU. The class attributes include an object CPU and its class is CPU\_control. The function IPL\_button will call initial function to start or restart the machine and return halt or not. If hault is 1 then it means something went wrong, and the display of MFR will change. The SS\_button will call function runsinglestep and return halt or not. Again, if halt is 1 then it means something went wrong, and the display of MFR will change. The function Load\_button will run the load instruction if the opcode of input is correct (000001) and return halt or not. If halt is 1 then it means something went wrong, and the display of MFR will change. The function Store\_button will run the store instruction if the opcode of input is correct (000010) and return halt or not. Again, if halt is 1 then it means something went wrong, and the display of MFR will change. The function LD\_button can change the value of each components according to the order of the GUI using the input from GUI. The function get\_number can return the value of each components according to the order of the GUI and it will be used to send the value to GUI.

**GUI:**

The GUI is built using the Java Swing Library, with individual UI components mounted on to a JFrame. The JFrame contains a JPanel which has the text fields representing all the registers on the UI.



The individual registers have a label, built using the component JLabel which is a display name for the register, and there’s a JText Field which is a text field used to display the contents of that particular register. There are a few load buttons next to some registers, built using JButtons.

Down below on the Panel have the space for entering individual instructions. The buttons with numbers 0-15 represent the 16 bit instruction set that can be entered bit by bit in the text boxes above. Each text box only takes 1 bit as input which can be either 0 or 1. No other input can be supplied

Down right on of the panel, there are 4 buttons: Load, Store, SS and IPL. The load button when pressed loads the instruction onto the specified register. Store is used to store the instruction to the memory. IPL button can be used to run the IPL.txt file and the SS button is used to run the instruction one after another as single steps.

The class PanelView.java contains a constructor PanelView which initializes the GUI. All the individual components have been added in this constructor. The class PanelView has a main method that initializes a frame and an instance of PanelView is called to show the UI on the screen. The Load and Store button functions have been handled using event listeners defined in the code that then trigger the backend to preform the specified opperations. The text fields have also been limited to only having binary numbers as input and output using action listener( KeyPressed and KeyReleased) methods defined in the code.

**Project 2**

In project 2, significant changes were implemented into the simulator, including a cache between CPU and memory, ALU and its condition code register, device 0 and 1 representing the keyboard and printer, many new instructions, and the algorithms implementation of program 1.

1. In design of cache, cache class inherits the memory class so the cache can read and write the memory directly. The cache is a fully associative, FIFO, write-through cache. It has 16 lines, and each line has a flag representing it is valid or not, cnts for FIFO count, and tag matching the block number in memory. Each line has 8 words so the tags are 8 bits long, and memory has 256 blocks.

2. The ALU has it a register storing the condition code, register Y to store the first number for computing, and the register Z to store the results. In the operation add, subtract, multiply, divide all regard the operation numbers or 2’s complement as unsigned numbers so if the result is less than 0 it would cause an underflow. The shift and rotate instructions can compute the associated signed numbers.

1. New instructions include :JZ, JNE, JCC, JMA, JSR, RFS, SOB, JGE, AMR, SMR, AIR, SIR, MLT, DVD, TRR, AND, ORR, NOT, SRC, RRC, IN, OUT. Their functions are all written in the CPU\_control.java.

4. The program1 needed a wide variety of new buttons and the textArea, so the user interface has been extended. In order to make the GUI and CPU link together to run program1, several functions have been added to the interact.java and PanelView.java, including program1input, loadprogram1, pro1target, pro1\_20numbers, printer in interact.java and loadProgram1, runProgram1ToTheEnd, input, and some changes in SS function in PanelView.java.

The details of program 1 instruction design can be read in Program1\_design.docx

**Project 2**

In project 3, the cpu added the trap instruction and trap table and trap entrance. The user interface is expanded for program 2. And the program 2 use the trap 1 to read 6 sentences and trap 2 to read target word.

The details of program 2 instruction design can be read in Program2\_design.docx

The utility of program 2 can be read in User Guide for Project3.pdf