Requirements Notes:

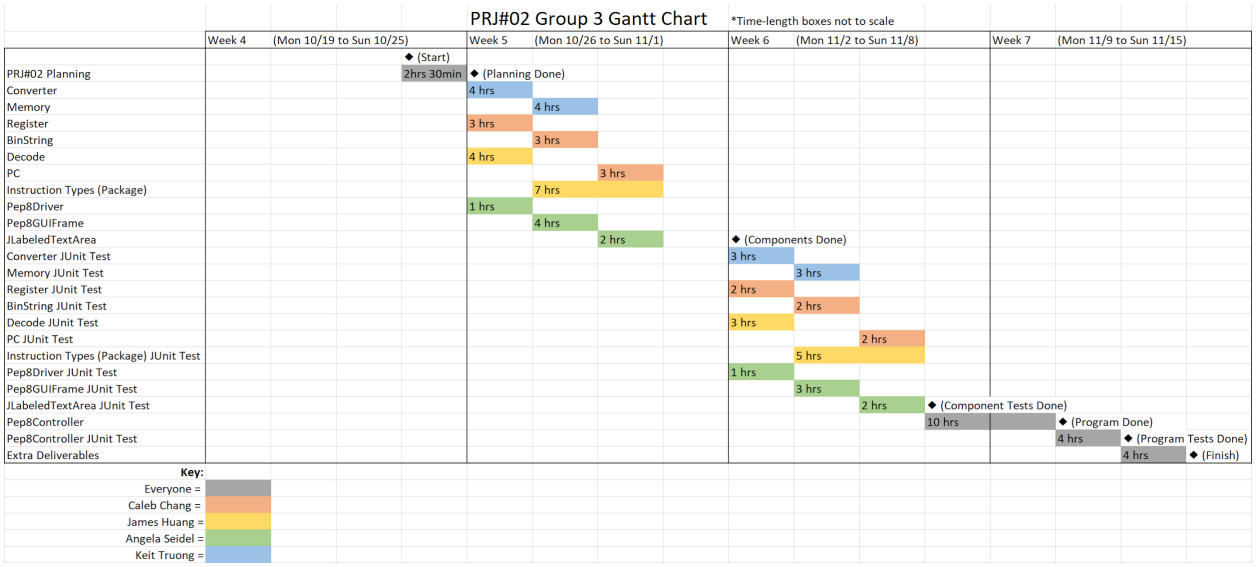
1. IntelliJ will be the IDE we use
2. Track version history with Git/GitHub
3. Program should work with assembly language, machine language, and binary
    a. Might work to have program that translates assembly to machine, then machine to binary and only worry about binary calculations
4. Run Source Code, Run Object Code, Clear Memory (Step Source Code, Step Object Code)
5. Add JUnit Testing

Weekly Plan:

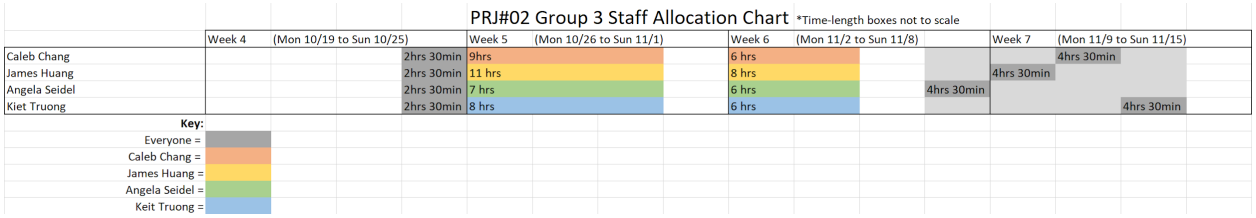
Table Calendar:

	Week 1	Week 2	Week 3	Week 4
Angela	PRJ#02a done planning	-) GUI Planning -) Complete Pep8Driver -) Complete Pep8GUIFrame -) Complete Pep8JLabledTextArea	-) JUnit Testing of GUI classes -) Maybe start working on Controller	-) Work on controller -) Work on controller JUnitTesting -) Work on any other extra deliverable requirements for final submission (demo videos, etc.)
Caleb	PRJ#02a done planning	Register, BinString, PC	JUnit Testing	Work on Controller
James	PRJ#02a done planning	Decode,instruction package	Junit testing ,controller	Work on Controller
Kiet	PRJ#02a done planning	Converter/Memory	-)JUnit testing assigned classes. -)*helping out wherever needed*	Work on Controller

Gantt Chart:



Staff Allocation Chart:



File Structure:

Model(Package):

- Description: contains the classes that keep and manage the status of data items.
- Includes the classes:
    - Converter (Kiet)
    - Memory (Kiet)
    - Register (Caleb)
    - BinString (Caleb)
    - Decode (James)
    - PC (Caleb)
- Instruction Types Package (James)

View(Package)

- Description: contains the classes that realize the program's (CLIor GUI) user interface.
- Includes the classes:
    - Pep8Driver (Angela) - Contains the main method that will open up Pep8GUIFrame and run the main program
    - Pep8GUIFrame extends JFrame (Angela) - Used to place the Java Swing elements in a nice layout for the user to see.

    - JLabeledTextArea (Angela) - Helper class for Pep8GUIFrame to make pre-formatted JTextAreas that have horizontal and vertical scrollbars as well as a headed title

Controller(Package)

- *Description:* contains the classes that implement all types of actions on the model (or data) classes to realize the operations initiated upon the requests of the user.
- Includes the classes:
    - Pep8Controller - Contains all the Model objects needed to carry out reading source and object code. (Should maybe have a run class)
- -NOTE: Can first create a step method that reads a single instruction, then create a run method that calls step repeatedly until step reaches a stop operation-
- (Completed collaboratively at end of program, used to tie everything together)

Extra Notes:
CPU has
InstructionRegister
Memory
ProgramCounter
Register

Model package

Converter

- Bin ---> Hex
- Bin ---> Dec
- Dec ---> Bin
- Hex ---> Bin
- Hex ---> Dec
- Dec ---> Hex

Memory

- Note: Could use an array to store memory. Translate a hexadecimal address to decimal and use that decimal address in the array to access memory locations from (0 - 6500).
- getDataAt
- storeAt
- getMemory
    - Returns array of memory consists of BinString

Register

- getRegister
- loadRegister

PC

- getPC

Decode

- decodeInstruction(String bin);
- Decode binary string and Return instruction obj

InstructionTypes package

Interface Instruction

- execute(unknown obj)
- getopcode()
- getOprand()
- getRegister()

Add implement Instruction
charin implement Instruction
charout implement Instruction
lw implement Instruction
sw implement Instruction
sub implement Instruction

View package

Pep8Driver

- Description: Contains the main method that will open up Pep8GUIFrame and run the main program.
- Fields: (none)
- Methods:
  - Public Pep8Driver() {
    - Placeholder constructor that does nothing when called
  - Public static void main(final String[] theArgs) {
    - Creates a Pep8GUIFrame object and sets it to visible, etc.

Pep8GUIFrame extends JFrame implements ActionListener

- Description: Used to place the Java Swing elements in a nice layout for the user to see.
- Fields:
  - // Strings for each Button label (to be used with the ActionListener deciphering which button was pressed)
  - // JTextFields for each field in the original Pep/8's CPU area
  - // JLabeledTextAreas for the input, output, source code, object code, and memory
  - The Controller object (so that the ActionListener can communicate to the Controller what to do)
- Methods:
  - public Pep8GUIFrame() {
    - Used to place Java Swing element in proper locations
  - private JButton makeButton(final String theButtonText) {
    - Helper method that creates a JButton that is already connected to an actionListener
  - private JTextField makeUnfocusableTextField(int theColumnLength)
    - Helper method that creates JTextFields with .setFocusable(false) so the user doesn't accidentally edit the CPU Register display.
  - @Override
    public void actionPerformed(final ActionEvent theEvent) {

        - Method that reads button inputs (by string id's to see which button) and triggers the right action by the correct button

JLabeledTextArea extends JPanel

- Description: Helper class for Pep8GUIFrame to make pre-formatted JTextAreas that have horizontal and vertical scrollbars as well as a headed title
- Fields:
    - private final String myTitleLabel
    - private JTextArea myTextArea
- Methods:
    - // Forward JTextArea methods so that they are accessible through this class)
    - Public void replaceRange(String theString, int theStart, int theEnd)
    - Public void insert(String theString, int thePosition)
    - Public void append(String theString)
    - Public String getText() (inherited from JTextComponent)
    - Public string setText(String theString) (inherited from JTextComponent)

Controller package

Pep8Controller

- Description: contains the classes that implement all types of actions on the model (or data) classes to realize the operations initiated upon the requests of the user.
- Fields:
    - // CPU display states as shown in the original Pep/8 program
- Methods:
    - Public void assemble() (makes source code into object code)
    - Public void load() (loads object code into memory)
    - Public void execute() (executes object code in memory)
    - Public void sourceCodeRun()
    - Public void objectCodeRun()
    - (maybe include below)
    - Public void sourceCodeStep() (can be called repeatedly for sourceCodeRun())
    - Public void objectCodeStep() (can be called repeatedly for objectCodeRun())
- NOTE: Can first create a step method that reads a single instruction, then create a run method that calls step repeatedly until step reaches a stop operation