# Group 1: Phase 4

Adam Mertzenich, Kritib Bhattarai, Michael Musa

November 3, 2021

## Contents

**1 Plan to Meet Minimal Requirements** **1**
1.1 Minimally Required Commands . . . . . 2
1.2 Implementation Plan . . . . . 2
1.2.1 Model . . . . . 2
1.2.2 View . . . . . 3
1.2.3 Controller . . . . . 3

**2 Going Beyond the Minimal Requirements** **3**
2.1 Other Commands . . . . . 3

**3 Graphical Mock-up** **4**

**4 Division of Labor** **5**

**5 Primary Point of Contact** **6**

## 1 Plan to Meet Minimal Requirements

In order to implement the minimally required commands we will make use of different design patterns. Below we describe a basic overview of how we will use the Model-View-Controller and Observer design patterns to allow different aspects of the application to work together towards our goal. We will start by designing our model so that all of the minimally required commands are able to be used and will then move forward to adding graphical views and controllers for the model which acts independently. This gives us a clear plan for moving forward by first implementing the basic commands in the underlying model before moving on to creating the graphical interface.

## 1.1 Minimally Required Commands

These are the minimal set of commands which need to be implemented for the project. Getting these working in our model will be the primary task during the beginning of the development process. Our plan for implementing these commands and how we will divide the work between our group is provided below.

- `aa IV` ⇒ Alter contents of accumulator to `IV`
- `ap MA` ⇒ Alter contents of program counter to `MA`
- `amb MA HB` ⇒ Alter contents of the byte at `MA` to unsigned hex value `HB`
- `ba MA` ⇒ Set a breakpoint at address `MA`
- `h` ⇒ Print this help message
- `mb` ⇒ Display all of machine memory as bytes in hex
- `n` ⇒ Execute next machine instruction
- `ob` ⇒ Display the portion of machine memory holding object code as bytes in hex
- `q` ⇒ Quit
- `r` ⇒ Run machine until error occurs or stop instruction is executed
- `s` ⇒ Display machine state (accumulator, program counter, etc.)
- `z` ⇒ Reinitialize program counter to zero

## 1.2 Implementation Plan

### 1.2.1 Model

In this project the model is/are the class(s) which represent a VM252 program. We have already implemented `encodedInstructionBytes`, `decodedInstructionComponents`, and `runProgram` methods as a starting point. These methods will be expanded or added upon in order to increase the model's functionality. Each command which we implement shall likely be a separate method to allow the view and controller to interact with the model in an orderly fashion.

The model will be an `Observable` class(s) which will allow it to be watched by the view and or controller respectively.

### 1.2.2 View

The view is principally an `Observer` of the model. The aesthetic design of the view can be developed independently from the model which will be supplied as it's developed. We will be making sure that the view is notified of all relevant changes to the model so that it is sure to be always up-to-date.

For specific view design plans, see the preliminary graphical mock-up.

### 1.2.3 Controller

The controller will also be an `Observer` of the model. Because the controller is intimately connected with the implementation of the model it cannot be implemented beyond aesthetic considerations until the model is complete. Focusing on completing the model will thus be the primary goal when starting out. Once the debugging process has been implemented then the controller will be developed.

For specific controller design plans, see the preliminary graphical mock-up.

## 2 Going Beyond the Minimal Requirements

Implementing the minimal requirements will be our first and primary goal. We hope to be able to implement many/most of these other commands but making sure that our graphical design and model functionality are smooth is the primary goal. With that in mind, we have made some considerations in regard to the other commands. We will keep in mind these commands when designing exactly how our model works so that minimal code rewriting will be necessary. For example, when designing the `mb` command we will write it so that it can be easily extended/modified to implement the `md` command. The best way of doing this will be to utilize the extensibility of classes so that to add additional features will would only need to extend/implement previous classes or interfaces.

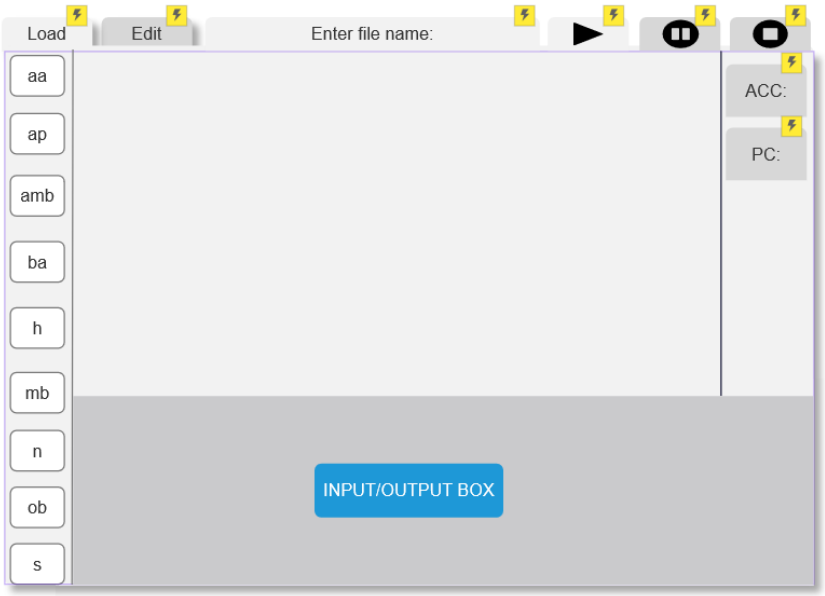
### 2.1 Other Commands

These are the other commands which we may implement following the minimal commands. Many of them are able to be created by simply extending previous commands.

- `amd MA SD` ⇒ Alter contents of the two-byte pair at `MA` to signed integer value `SD`

- `amdx MA HD` ⇒ Alter contents of the two-byte pair at `MA` to unsigned hex value `HD`
- `bl L` ⇒ Set a breakpoint at source-line `L`
- `cb` ⇒ Clear all breakpoints
- `md` ⇒ Display all of machine memory as 2-byte data in hex
- `mi` ⇒ Display all of machine memory as instructions, data, and labels
- `od` ⇒ Display the portion of machine memory holding object code as 2-byte data in hex
- `oi` ⇒ Display the portion of machine memory holding object code as instructions, data, and labels
- `t` ⇒ Toggle instruction tracing

# 3 Graphical Mock-up



## 4 Division of Labor

In order to divide the work on the project fairly we have split up various tasks so we are able to work individually before combining our work. First we have split up various minimally required and extra commands to each member of the group. Once we implement those aspects of the model we will then divide the work of creating the view and controller by having us work on the portions relevant to the parts of the model we developed. When necessary two, or all three, members of the group will work on implementations when there is overlap. Below we have a *general* outline of which commands we currently plan on each implementing, with the acknowledgement that some of this will change as we figure out the finer details.

1. Kritib Bhattarai
    - `ob`
    - `r`
    - `q`
    - `s`
    - `c`
2. Michael Musa
    - `ba`
    - `h`
    - `mb`
    - `n`
    - `bl`
    - `cb`
3. Adam Mertzenich
    - `aa`
    - `ap`
    - `z`
    - `amb`
    - `amd`
    - `amdx`

## 5 Primary Point of Contact

Adam Mertzenich will be the primary point of contact for our group. You can email him at mertad01@luther.edu to ask questions.