Georgia State University

Mano’s Computer Simulator

James Wingo

Computer Architecture

Dr. Belkasim

**1. Introduction**

Manos’ Computer that is introduced in his book Computer System Architecture is a powerful learning tool for understanding how the basic design computers work. Though it is a theoretical computer that does not actually exist, it exemplifies some of the most important concepts that nearly all computers use. Learning how it works is essential for understanding the fundamentals of Computer Science. Because of this fact, a Java Application called Manos’ Computer Simulator (MCS) was made in order to facilitate learning the basic concept of the computer. The purpose of this paper is to give an outline of the goals of the app and a brief description of the computer. It will then discuss how the MCS works from the user’s perspective and then give overview of the programing design of the MCS.

**2. Goals and Preliminaries**

The main goal of the MCS is to interactively show the computer that is outlined in Chapter 5 of Computer System Architecture[1] and how the computer changes over time depending on the starting state of the computer. A few secondary goals were also achieved, while working on this project. First, the code for the MCS was designed to be as modular as possible. Aside from good programing practices, this goal was set in cause a student wanted to extend the MCS to be used in their own project. Also, the structure of the program was designed to adhere as closely to the structure of the computer as possible. The reason that this goal was set was so that if a student were to look at the code and cross reference it with the book’s description they would find that even internally the MCS is similar thereby making a more multifunctional learning tool.

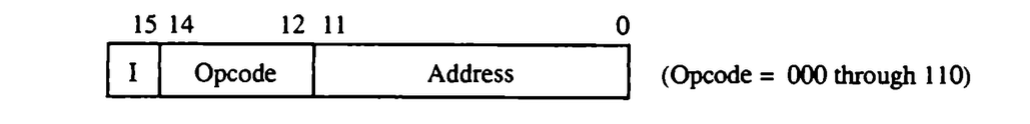
The MCS was designed to be as self-explanatory as possible for a student who is learning how the computer works. However, there are a few things that the student must be familiar with before using the MCS. Anyone who is trying to learn how the computer works should be familiar with all of the materials covered in the book Computer System Architecture up to Chapter 4, which includes but is not limited to the behavior of buses, registers, memory and decoders. The user will also benefit greatly from knowing the register transfer language that Manos describes in his book.

**4. Description of the basic Manos computer**

A brief description will be given of the computer to clarify the terminology used in the MCS. Manos’ computer is physically consists of registers. Each of these registers has a different purpose within the computer and holds information pertaining to their functions at any given time. These registers are used for processing data that is in a memory unit (MU). There are 6 additional registers that are general purpose. The first is the Address register (AR), which holds an address to the memory. Next is the Program counter (PC) that holds an address to an instruction that is in the program memory. A Data Register (DR) is used by the computer to hold an operand from memory. An Accumulator (AC) is present in the computer in order to hold a value that is going to be processed by an adder logic circuit that does basic functions to the data passed to it. An Instruction Register (IR) also holds an instruction code that will be processed. Finally there is a temporary register (TR) that is a general-purpose register that can be used to store any value.

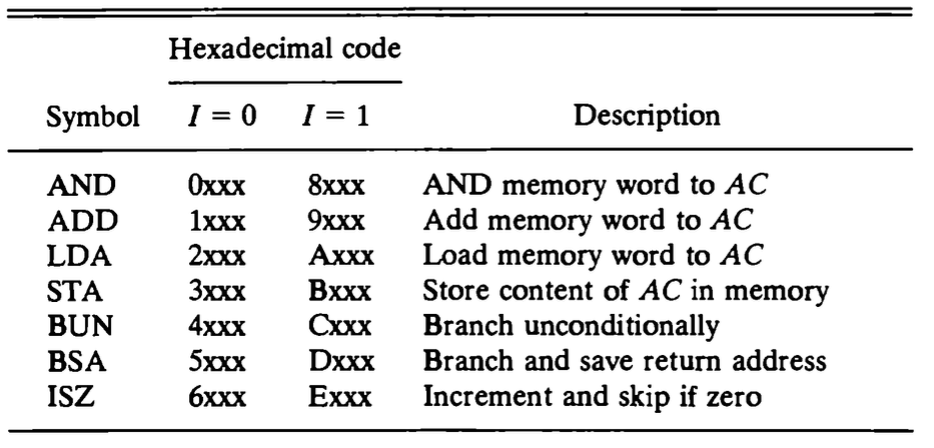
All of these registers are connected by a bus system where the bus is controlled by a selector switch whose inputs are S2S1S0, where each Sn is a binary value. Depending on the S values, a register is picked. Each register has a load bit that allows it to pick a value off of the bus. The only exception this is the AC, where its input is calculated based on the DR’s input and the Adder and logic circuit.

A Register that can hold an instruction can be formatted in one of 3 ways described in Manos’ book. However, the MCS can currently only do Memory-reference instructions, which are in the following format shown in Figure 1.



**Figure 1**. Instruction Code.

The “I” in the instruction code indicates indirection. If the “I” bit is set to 1 this means the Address in the instruction contains the address of the command. The address in the instruction code is the value in binary of the address in the MU and the Opcode section of the instruction signifies what command will be done. To clarify the Opcodes, a reference table is shown below where “x” can be any hexadecimal value.



**Table 1**. Memory reference instructions

Each register is connected to a clock and no command will be executed without the clock cycle. Each clock cycle is designated through time with the notation T0, T1, T2… etc. Having a clock labeled like this means that every command sequence can be notated through an interval of time.

For every command an instruction cycle the computer goes through the following steps.

1. Fetch: Fetches an instruction from memory
2. Decode: takes the instruction and decodes it for further evaluation
3. Read: Reads from memory if indirection is required by the instruction
4. Executes the instructions

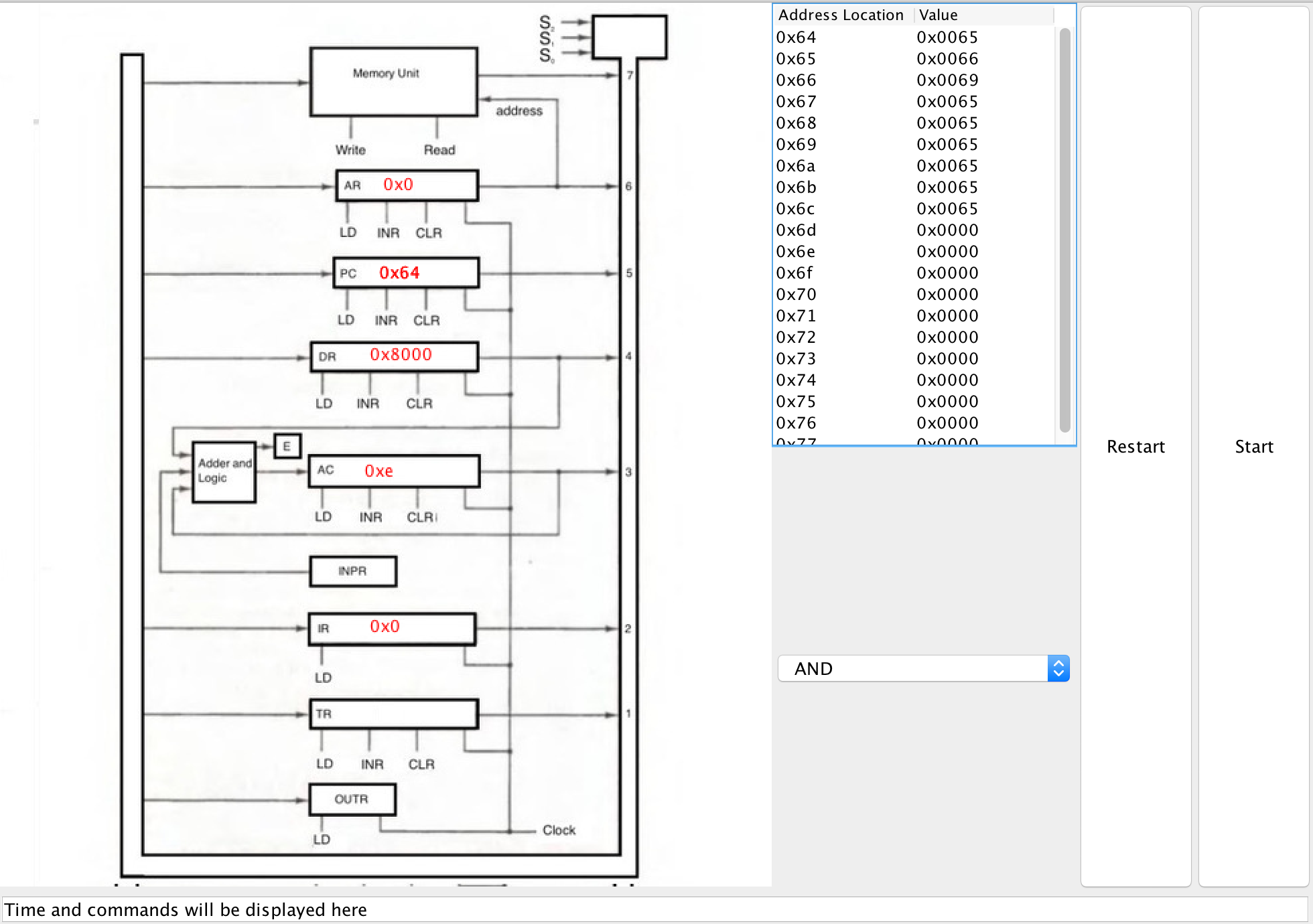
As one can see, every command requires at least these 4 steps, which will be designated along with the clock cycles T0, T1, T2…

From here, the instructions take place and this is where the MCS can be extremely helpful for visualizing the instruction cycle, including the execution of all of the memory reference instructions. With this minimum knowledge, ideally a student will be able to use the MCS to have a general understanding of how the computer works

**5. Application Description**

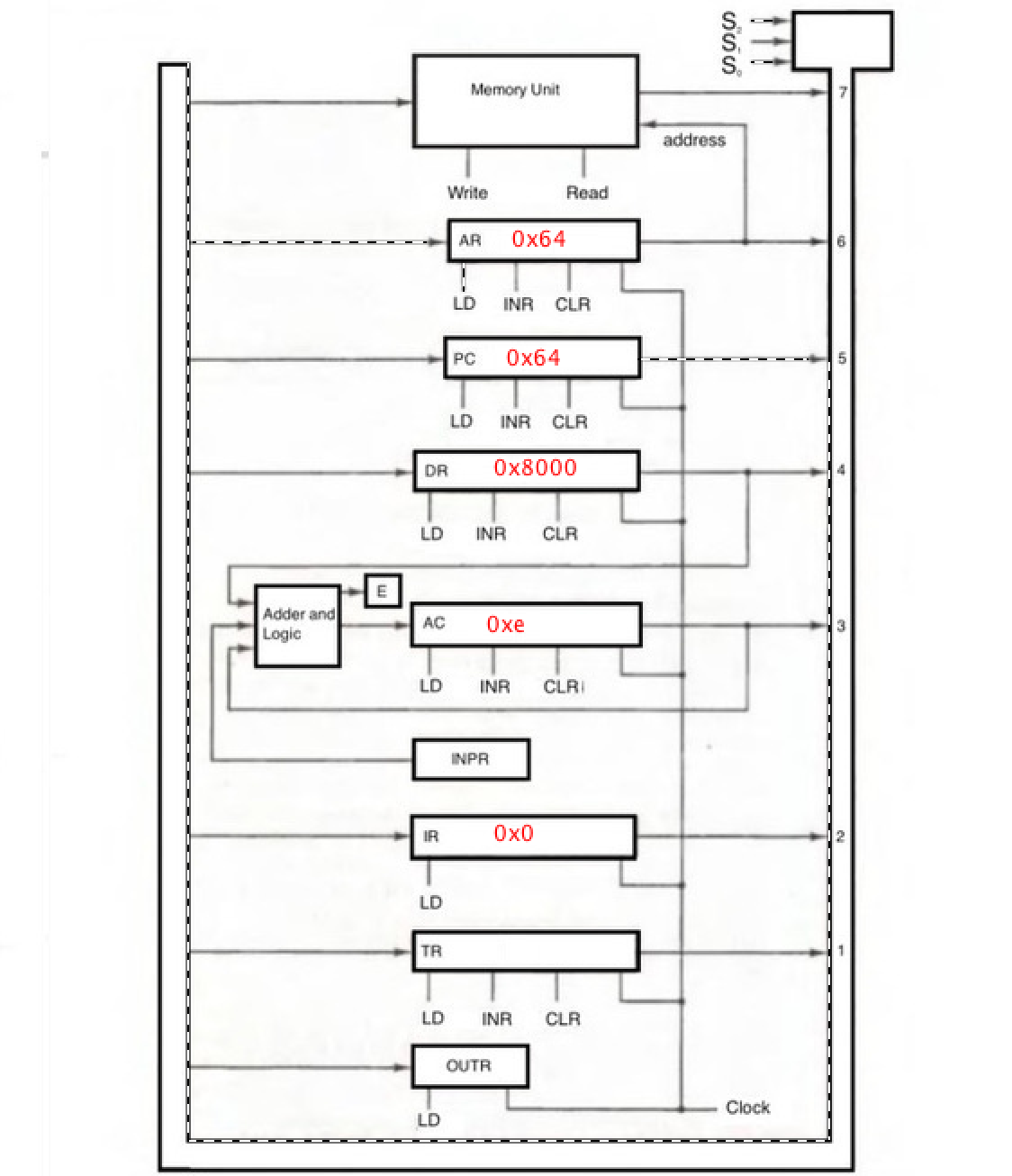
At this point a user should be able to open the Java applet and be able to use it to begin learning how the computer behaves at every time interval. However, a description is given here to clarify what each part of the MCS does.

For reference, a screen shot of the MCS is shown in Figure 2 below.



**Figure 2.** MCS screenshot

On the left, a user will see an exact copy of the Manos computer layout that is in the book. Using the exact same picture was an important decision. This image is what a student would have seen, while reading the book or while taking a class based on the book. Having this image cuts back the amount of confusion, while using the MCS. Within each register is the hexadecimal value of the contents of the register. The register values are not editable directly in this view. As each transfer occurs, an animation will be displayed on the image indicating which bus lines are being used, and the direction that the data is traveling. The MCS will also show which inputs on each register are selected. Also, the selector switch for the common bus also coincides with the register that is transferring data. Figure 3 shows the MCS while it is actively running. Though the direction cannot be shown here, it gives a clear indication of what the MCS can do as far as what it is able to display while it is running. Note, the clock line is not shown to be active. This behavior was done on purpose. When the clock line is displayed, there is too much animation on the MCS and it can be a bit distracting. So, it was left out.

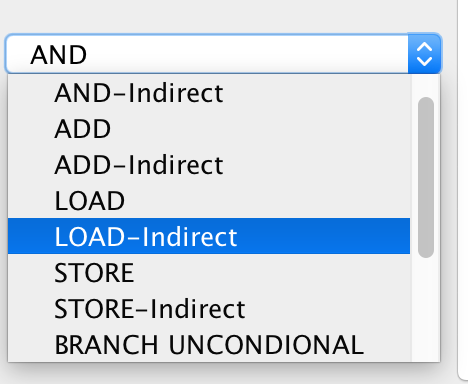


**Figure 3.** Active Computer

The table in the center of the app is a list of memory location and their associated values. This table is representative of the memory area in the memory unit. Note that the memory area listed here is much smaller than what is specified in Manos’ book description of his computer. A smaller memory was chosen so that it would be easier for a new user to use the tool without getting overwhelmed with the amount of data. Also, The table is pre-populated with values so that the computer can be run immediately when the MCS starts. However, a user can directly change the memory area by double clicking on any of the cells in the table. All data in this view is shown and should be entered in hexadecimal format.

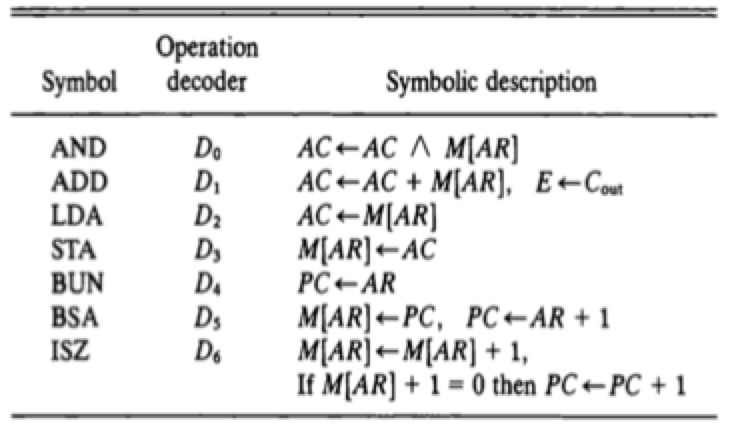
To the right of the table are two buttons labeled start and reset. The start button simulates a clock pulse and executes a single time interval of the MCS. Because of the way the clock works, the commands will need multiple start presses to go through an entire command cycle. The reset button will move the computer back to an initial state and can be used to “interrupt” a command cycle. This reset function is not part of the actual computer. The button was simply added for usability in case a user wanted to start over.

Below the memory table is a combo box selector, which contains preset commands that the user can choose from (shown in Figure 4).



**Figure 4.** Combo box.

When a user chooses a command the registers and the memory unit get repopulated with the correct values so that the selected command will be executed. Table 2 shows a list of the commands a user can choose. The symbolic description indicates the final desired result. The user can also choose an “indirect” version of these commands, which show how indirection is used in the commands. With this combo box a user would not even need to know the opcode values to under stand what is happening step by step.



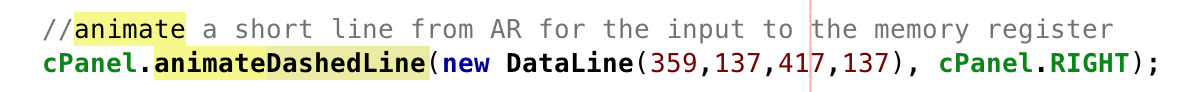
**Table 2.** Command list

At the bottom, of the MCS a user can see the current state of the computer. This box will display all of the register transfer information at each Clock pulse. The output box will also tell the user when a clock cycle isn’t used with a descriptive message.

**6. Overview of programing methodology**

Going into a line-by-line description of the code will not be covered here because of the length of the application. However, the overall design methodology will be discussed and the classes and their methods will be discussed. So, if someone were to want to look at or change the code, they would be able to understand the design choices.

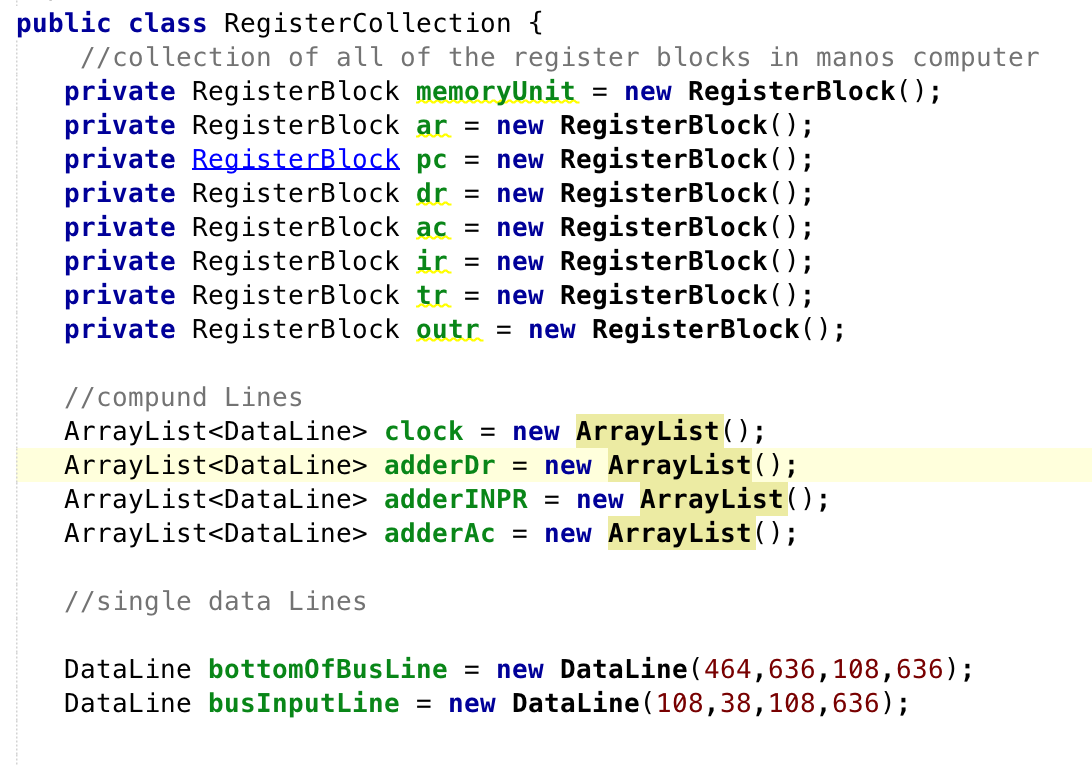
At the base level, the panel that displays the image of the computer is a jpeg image. All of the changing information is drawn on top of this image. To do these animations, a ColorPanel Class was created. This class is an extension of java’s Jpanel Class and gives methods to allow a programmer to easily draw colored pixels, draw texts and animate a dashed line. The animated dashed line method takes in two pairs x and y coordinates that are part of the dataline object (the details in which will be discussed later), which represent the start and end of the line. The AnimateDashedLine method can also take in a second parameter, which takes in one of four pre-defined constants. The constants are LEFT, RIGHT, UP and DOWN. This parameter is used to choose the direction in which the data “flows” in the animation. An example on how to use this method is shown in Figure 5. Note, that this call creates a new reference of DataLine but this isn’t required. The programmer could pass a pre-existing Dataline object



**Figure 5.** Animate Dashed line example

One of the goals of the MCS code was to adhere to the internal design of the computer as closely as possible. Since the computer is physically made up of registers and bus lines, the code reflects this by making each of these elements their own class. First a DataLine class was created. This Dataline is a simple class that holds the pixel location of where the data line starts and ends. These lines are then drawn using the ColorPanel object.

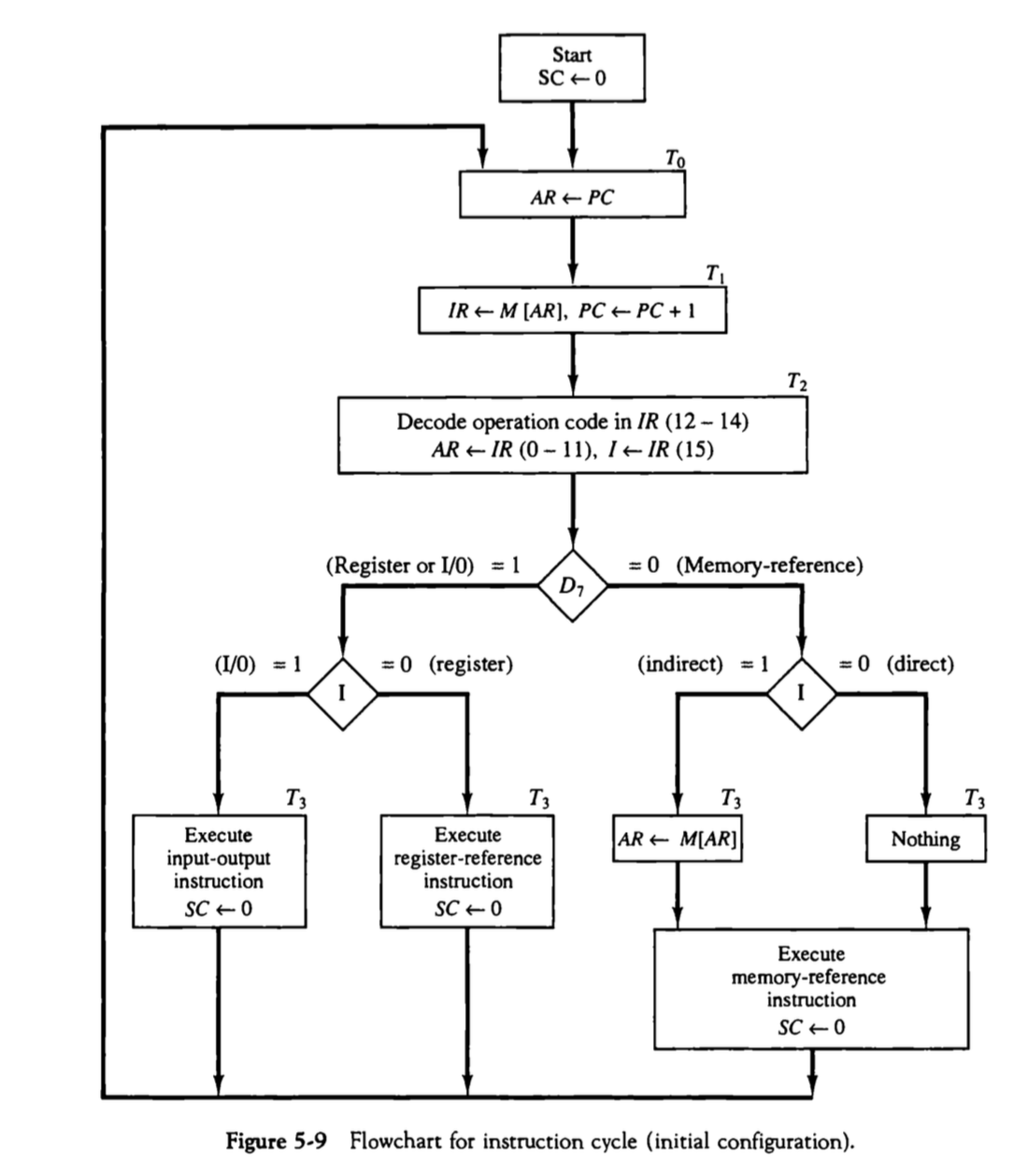
Each register is an object of a newly defined register class. The register class contains a register’s current data as well as all of the datalines that connect to the register. These registers and datalines are then put into a RegisterCollection class. This class could be viewed as the skeleton of the computer. The elements are all there but there isn’t any logic to run the functions. In this class, all of the register objects are initialized with appropriate values. This includes the pixel locations of each object in the computer. It also includes the data lines that are not included in the register. For instance, the adder and logic circuit is not a register so it’s Dataline objects are defined here. Figure 6 shows all of the objects that belong to this class for reference.



**Figure 6.** Register Collection object declerations

In order to get the MCI to actually behave like the computer another class was created called ManosGUI. This class contains a single object of the RegisterCollection class. The first thing that this class does is set up all of the Jframes and windows with pre set sizes and names. It then initializes and launches the frame and its listeners using and initComponets method. Listeners are needed for the combo box, the table and both of the buttons. One important thing to note is that the combo box listener contains a smalls state machine where depending on which element is selected a hardcoded value is then assigned to the registers and to the memory table. Also, another listener worth mentioning is the start button listener. When the start button is pressed, a sequence of events always happens. First the computer image is cleared, all of the registers are then updated and the a meathod called runManosComputer is called, which will be discussed next.

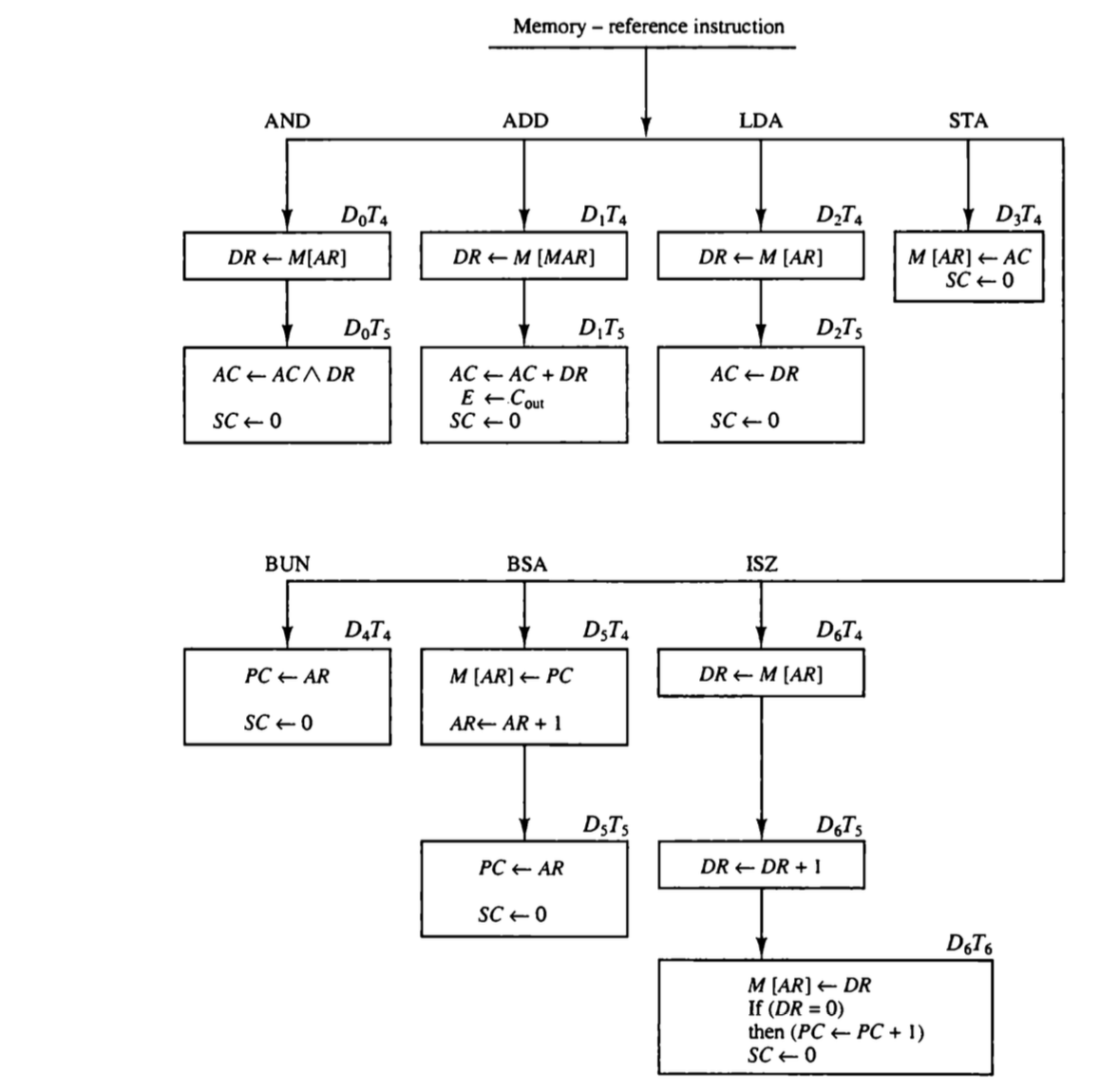
The runManosComputer method is what could be considered the “brains” of the computer. Just like the rest of the design, this logic is based closely with how the computer would have behaved if it actually existed. To do this, a state machine was created using a java switch statement. Every time that the runManosComputer method is called the currentState is evaluated and then the comptuer is evaluated at the corresoponding case. Then, the action required by that state is done.Finally, the next state is decided based off the following flowchart taken from Manos’ book.



**Figure 7.** Flow chart for state machine.

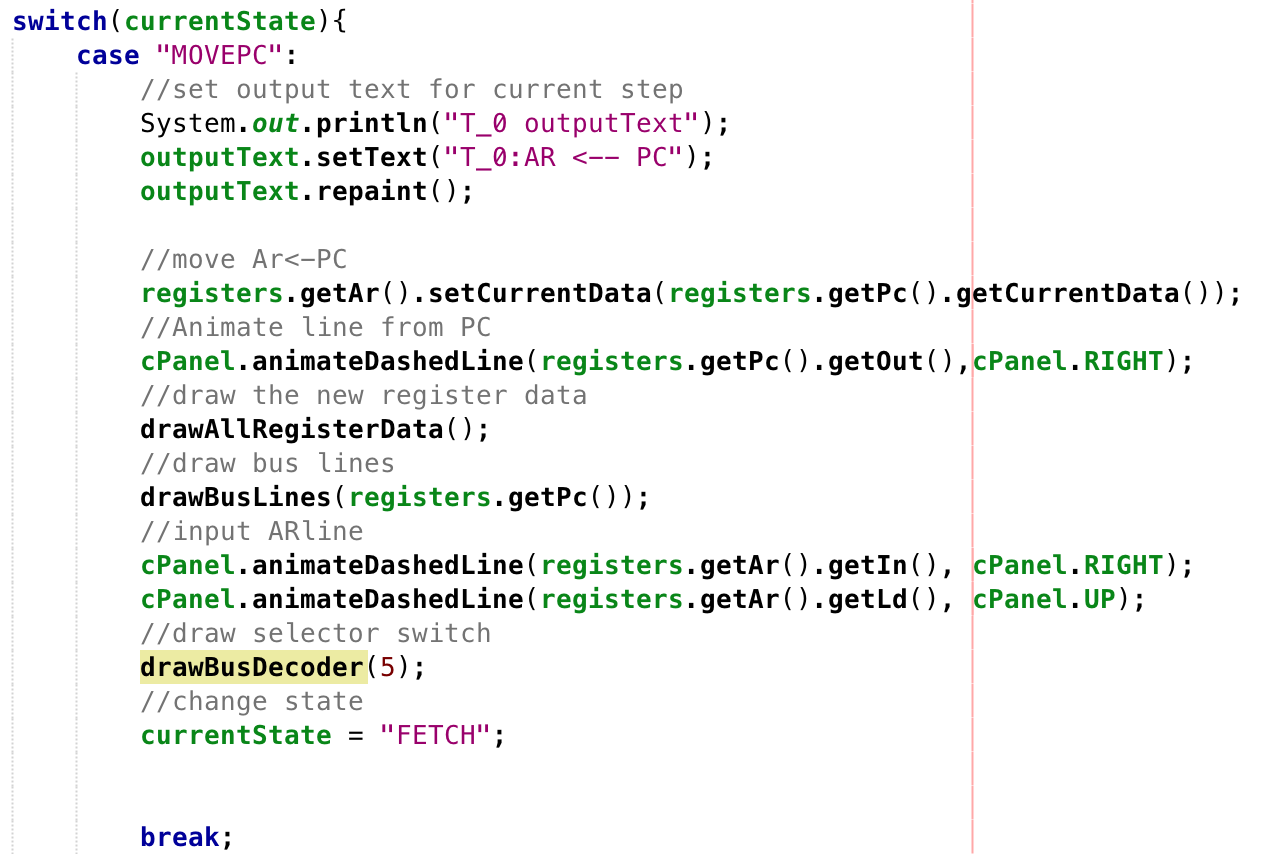
For the first three time steps, the switch statement follows figure 7. (which is Figure 5-9 in manos’ book) If the code is traced, one would see that each state follows the decision blocks of the flow chart.

Figure 8 shows the flow chart that the rest of the instruction. Again, the switch case statements follow this chart and starts evaluation immediately after T3 in Figure 7.



**Figure 8.** Flow chart for memory reference instructions.

Each state not only updates the register content but also sets the output text for the user to see. It then animates the corresponding buses and updates the computer register so that the new value is displayed on screen. Finally, it draws the selector switch and updates the current State to the next state for the next clock cycle. An example case is shown in Figure 9. All of the states adhere to this format.



**Figure 9.** Example state from runManosComputer.

Note that drawBusDecoder is a simple method that draws the inputs to the selector switch. The input is an integer that corresponds with the number on the bus.

From here, it is as simple as creating a manosGUI object in the main method to run the MCS.

**Conclusion**

With this information, one should be able to understand not only the overview of how the MCS works from the user perspective but also how it works internally. With this information, someone could come and use this base program to expand it into being able to accept input and output instructions or use it to perhaps show how pipelining works. Though Manos’ computer is a simple design it is something that is a perfect starting point for learning the basics of computer architecture.

The project is distributed using a java .jar file. To run on a computer with java installed one simply needs to double click this file and the application should launch. Source code can be found at <https://github.com/jwingo3>.

Resources

1. M. Morris Mano “Chapter 5- Basic Comptuer Organization and Design,” in *Computer Arichtecure,* 3rd ed. Prentice Hall pp 123-164