Section 1 - Project Description

1.1 Project

Prometheus

1.2 Description

Prometheus is a project to create a MIPS Emulator and give a visual representation of the processor. This software was created due to the lack of a modern MIPS emulator software. It was created for school credit as a directed study under Dr. Brad Barnes at the University of Georgia. Dr. Barnes taught a computer architecture class and utilized a similar but outdated software called, “Pathsim,” and had difficulty getting all the computers to run the software. By using this software we hope to achieve a better teaching/learning experience by creating this software which is compatible with modern systems as well as add additional features, input from text file, support for different architectures, and extensibility for future development.

1.3 Revision History

Date	Comment	Author
August 25, 2015	Begin creating the software requirements	Frank Hu
Decenber 11, 2015	Software Requirements Mostly Finished	Frank Hu

1.4 Reviewers

Name	Position
Dr. Brad Barnes	Professor, Department of Computer Science, UGA
Mr. Michael E. Cotterell	Professor, Department of Computer Science, UGA
Nick Klepp	Graduate Student, Department of Computer Science, UGA

Contents

.

Section 2 - Overview

2.1 Purpose

The purpose of this software is to give students in future computer architecture class a more interactive and beneficial learning experience through the use of this software.

2.2 Scope

Project Prometheus will be developed to provide an inner look at a MIPS processor. It will go down to the physical wire layer. The user will be able to transition between the layers of the processor to full see what is happening. It will display the values of each wire, as well as the names of the hardware. It will provide animation of the wires that will represent their values as well and to show the flow of the processor when it is running. The user will be able to input data into the registers directly as well as be able to load MIPS commands into the software. The software will give the user the ability to also input data into the Data Memory. Project Prometheus will be utilized by professors to teach class as well as computer architecture enthusiasts.

2.3 Objectives and Success Criteria

Our final system ("Project Prometheus") should be easy to use, easily maintainable, reliable, and favored by both professors and students wishing to learn about how computer processors work.

2.4 Requirements

Your mileage may vary -- we typically break down the requirements to provide a ballpark estimate.

2.4.1 Estimates

#	Description	Hrs. Est.
1	Brief description of task / module with link	4
2	User Cases	20
3	UML	20
4	Sequence Diagram	20
5	Future Work	1
	TOTAL:	65

Section 3 - System Architecture

This document defines the entire base system architecture of Prometheus. As of right now there are no modules to plugin to Prometheus, however it has been speculated to that there may be future work to integrate plugin architecture.

Section 4 - Data Dictionary

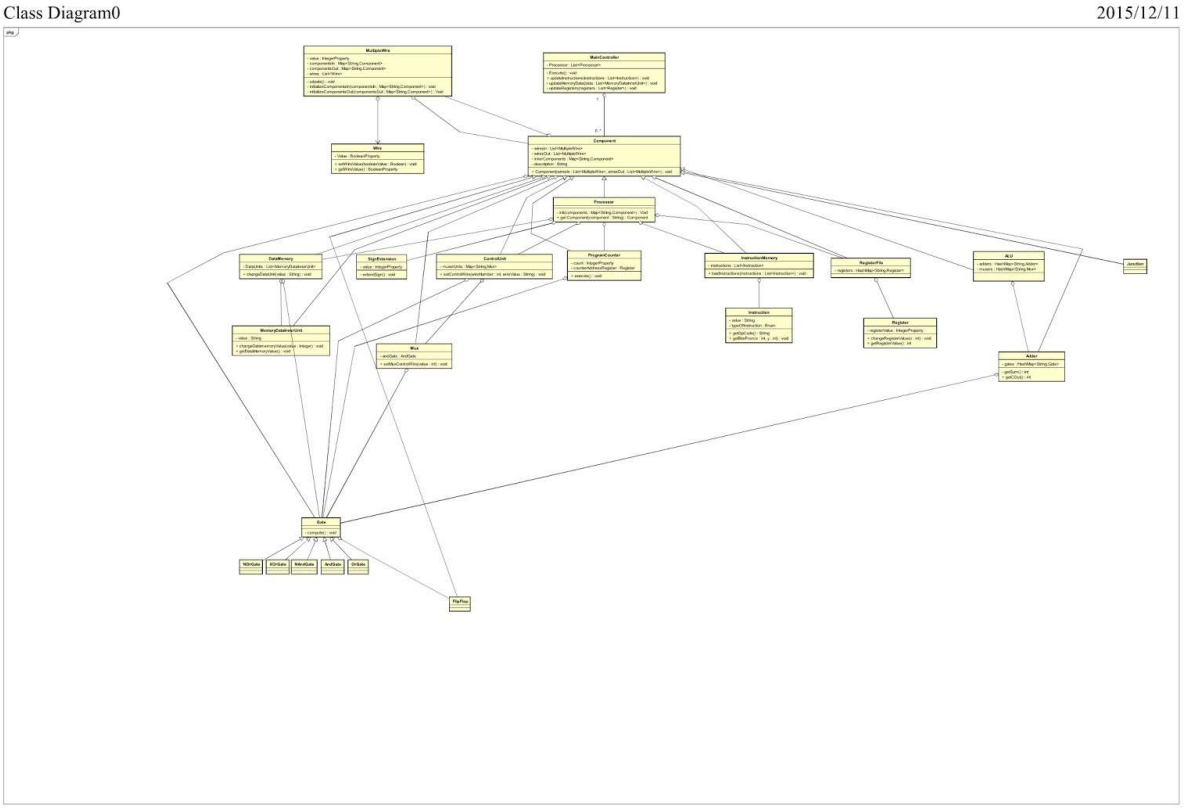
http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html
http://courses.cs.washington.edu/courses/cse378/09wi/lectures/lec08.pdf

Section 5 - Software Domain Design

5.1 Software Application Domain Chart

5.1.1 UML

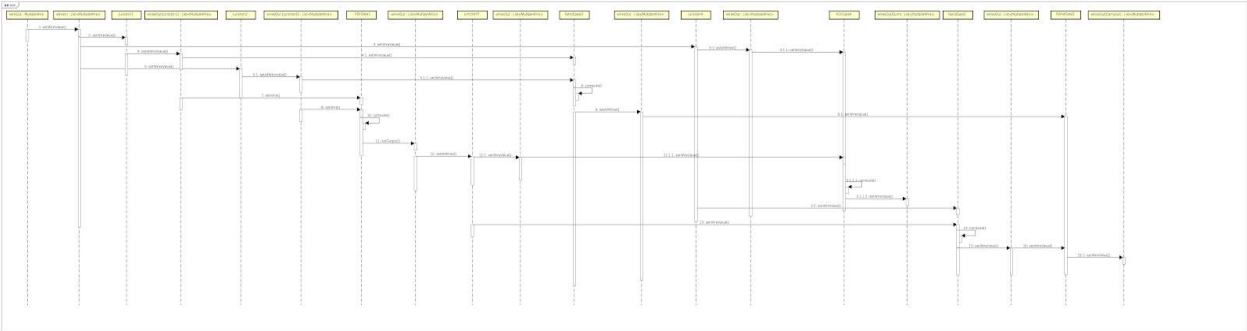
Each component in this UML will extend the Component class. Each Component lower on the tree, displays a lower level device, which typically will be aggregated to the component above it. If it is not a component aggregation, it is a data structure that is an aggregation of the component above it. The gate class is there for simplicity, it is saying that these devices will have gates, however in order to make the diagram smaller, I've made it so that the devices only have the aggregation "gate" as opposed to multiple aggregations of 1-4 different types of gates.
For a closer view please visit :
https://drive.google.com/file/d/0B-25HuV1bLtTSHhQbW10a1U3OHM/view?usp=sharing



5.1.2 Diagrams of Adder

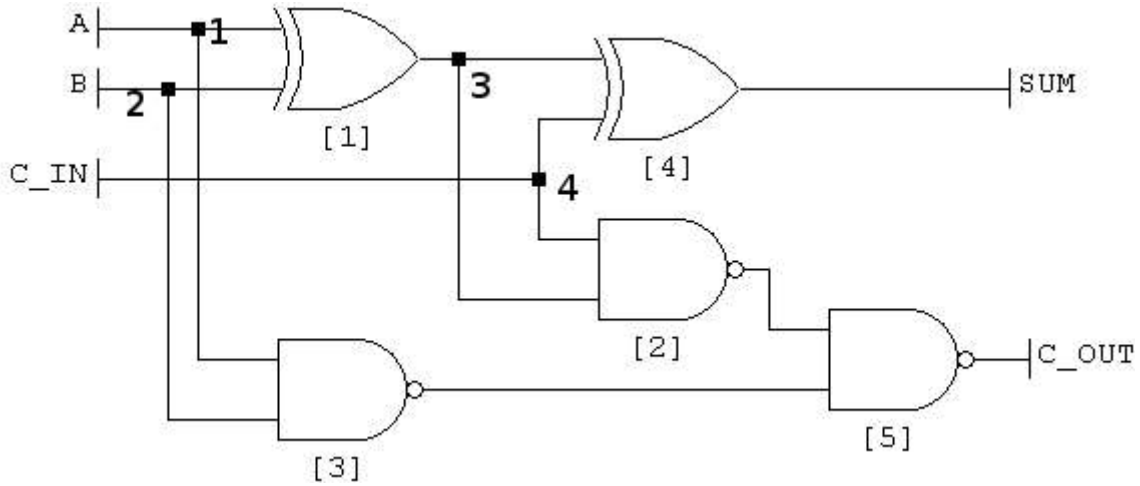
5.1.2.1 Sequence Diagram of Adder

Below is the sequence diagram for the adder. It displays all the methods that are called and what is happening as the two inputs A and B and C_IN are changed.
For a closer view please visit :
https://drive.google.com/file/d/0B-25HuV1bLtTUE9SVzBMTFowbTg/view?usp=sharing
An explanation along with actual code is at this link :
https://docs.google.com/document/d/1xtlTVOJYHYZZhOhjLvY2JhFL7oMM3Y3tw7sNTaJqbJI/edit?usp=sharing



5.1.2.2 Visual Diagram of Adder

This diagram of the Adder displays the gates and junctions. The junctions are the black dots, and the gates are displayed in accordance to standard procedure of Gate protocols.
For a closer view please visit :
https://drive.google.com/file/d/0B-25HuV1bLtTQzZNLUZRX1Bham8/view?usp=sharing



5.2 Software Application Domain

The software design is broken up into three separate parts. There is the component part, the User Interface(FXML/JavaFX Implementation), and then a utility part that is essentially where everything that doesn't fall into the first two is put.

Components are : Processor, ProgramCounter, ControlUnit, SignExtension, DataMemory, InstructionMemory, ShiftLeftTwo, RegisterFile, ALU, Instruction, Register, Adder, Mux, MemoryDataUnit, FlipFlop, Gate.

These components all extend the Component abstract class, which defines what each device should have.

Each device will be made up of either Sum Of Products or Product Of Sums implementation. From developing a discrete mathematical representation of each device, we can create the gates composition of each device. Since we know the gate composition, each device's output can consists of the input going through the multiple gates, emulating an actual processor. This effectively makes Prometheus an emulator for a MIPS Processor.

The wires in between each device are all multiplewires. Even the MultipleWires that only consist of one wire will be multiple wires. This is to provide consistency between each view between each interface, we will display multiple wires at the high level, and wires are only displayed when looking at a multiple wire.

This logic is created by the idea that a wire is typically more than one wire. I.e. a wire that has a value of 302, actually is represented through wires that display the binary version of 302. By this logic, a wire that is represented as "one wire" in the architecture, actually is more than one wire, misleading many. By implementing MultipleWire and letting the user click on a single wire and showing them that a multiplewire has many wires, will help clear up how the MIPS processor actually works.

Processor is an extension of Component is by the logic that we may have more than one processor one day. By making it a component, (Which abstractly it is), we can implement more than one processor easily.

A Comprehensive high level description of each domain (package/object wherever it is better to start) within the scope of this module (or within the greater scope of the project if applicable)

5.2.1 Components

This is the abstracted concept of each device within a processor. Each class should have a hashmap of wires in. The hashmap should have keys of "input1", "input2", for each respective multiplewire that defines the input for input1/input2. There should also be a hashmap of outputs, with similar keys that define the output of the device. The innerComponents variable is meant to map each name of each device to a hashmap, The nameOfComponent, and Description are meant to define the respective class and will be utilized later on in the GUI part.
A high level description of the family of components within this domain and their relationship.

Include database domain, stored procedures, triggers, packages, objects, functions, etc.

5.2.1.1 Mux of Components

Muxer is a makeup of many different gates which perform the mux function.

5.2.1.1.1 Do Mux Process of Mux of Components

This Mux Process should do the process of muxing according to the output. This process should be done through each of the

5.2.1.2 Processor of Components

The processor is typically what contains all the devices. We will monitor what's going on inside of the processor. In order to utilize the load data, change instruction, and change register functions. The processor should have these functions available for the the button click changes.

5.2.1.2.1 Change Data memory of Processor of Components

This processor should be able to change the data memory that is an innerComponent of the processor.

5.2.1.2.2 Change Instruction of processor of Components

When the user wishes to load new instructions into the processor, the processor should be able to change the instructions to it's inner components. These instructions are abstracted for later GUI feasibility.

5.2.1.2.3 Change Registers of Processor of Components

When the user wishes to change the values in the registers, the changeRegister function should be callable to replace ALL the register values with the new register values.

5.2.1.3 Control Unit of Components

The control unit has multiple outputs to different places. It needs to process it's input in order to determine the outputs.

5.2.1.3.1 Process of Control Unit of Components

Process the inputs in order to determine the outputs. From there each output should be changed accordingly in order to compensate for the instruction processed.

5.2.1.4 Sign Extension of Components

This unit does not do anything. It is sort of a larger junction, it simply adds more wires to the wiresOut, not changing the output value, but simply adding 0 bit wires to the output.

5.2.1.5 Data Memory of Components

Data Memory is where all the data memory is held. These values are not being
since not enough research went into learning the Data Memory unit. We are abstractly

representing the data memory with DataMemoryUnits, and using these units in order to represent the data memory.

5.2.1.6 ProgramCounter of Components

The program counter represents the place where the program begins it's next step. The only option should be run next instruction/next step where it runs the next instruction it has.

5.2.1.6.1 RunInstruction of ProgramCoutner of Components

Runs the next instruction.

5.2.1.7 InstructionMemory of Components

InstructionMemory like data memory is not being represented here. We have included an instruction data structure, however this data structure is only used for the GUI, so that the user can see which instruction is currently running and for ease of use when creating the interface.

5.2.1.8 ShiftLeftTwo of Components

ShiftLeftTwo of components has a binding where it will shift left two the values. This will be implemented through gates, the inner components should do all the work, so there is no inner method that will do << of some sort.

5.2.1.9 RegisterFile of Components

The register file holds all the registers. Registers are not being represented due to lack of research. We will be utilizing the register data structure within the register file and abstractly implementing all the functions.

5.2.1.10 ALU of Components

ALU processes values. Since all the devices inside should provide the correct output, no abstract methods are required. We only require the basic abstract component fields.

5.2.1.11 Adder of Components

Adder adds two bits together and has an output of sum and C_out. These two values are computed through the devices within it, so we don't need to abstract these methods out either.

5.2.1.12 Gate of Component

The four gates will be shrunk into this entry in the software document. There are five gates: Nor, XOr, Nand, And, Or. These gates should have two inputs and one output. The bindings to each of the gates should correspond to their respective functions.

5.2.2 MultipleWire
MultipleWires are wires that connect one device to another device. The regular wire class is only used when a user wishes to zoom in on a wire, which the gui will then take each wire and determine how many wires to display, and their respective values.

5.2.3 Wire
The wire class represents the 1 bit wire. This is used for the GUI when zooming in on a multiple wire.

5.2.4 Instruction
The instruction class is used for the GUI. This gives the easy access to the instructions and a workable format to display what the instructions are in the current processor.

5.2.5 MemoryDataInnerUnit
This data structure represents the data memory. Since we are unable to determine the Memory Data's inner workings, we abstract the memory data to a data structure. This data structure in turn is used to implement what the DataMemory device actually does.

5.2.6 Register
Like the DataMemory, the RegisterFile is a device that we are unable to implement. The register in turn is used to help perform the RegisterFile's duties by abstracting the concept of a register file out and implementing it without emulating it.

Section 6 – Data Design
No external Data, except for the processor. This processor will be a serialized java object that can be exported by a file option. Everything within the processor should be saved and should be importable to another prometheus program.

6.1 Persistent/Static Data
The only persistent data in Prometheus is the saved processor, we will not maintain a database.

6.1.1 Dataset
No Dataset.

6.1.2 Static Data
As of right now, there are no static data members in Prometheus.

6.1.3 Persisted data
Persisted data is the processor. One can save the processor in the current state with the instruction memory and registers preloaded, this can easily be done by serializing the processor and then saving the java Object.

6.2 Transient/Dynamic Data
No Transient/Dynamic Data

6.3 External Interface Data
Since the Persisted data is a serialized java object, we can simply import the java processor to look at.

6.4 Transformation of Data
No external data transformations

Section 7 - User Interface Design

7.1 User Interface Design Overview
Pictures, high level requirements, mockups, etc.

7.2 User Interface Navigation Flow
Diagram the flow from one screen to the next

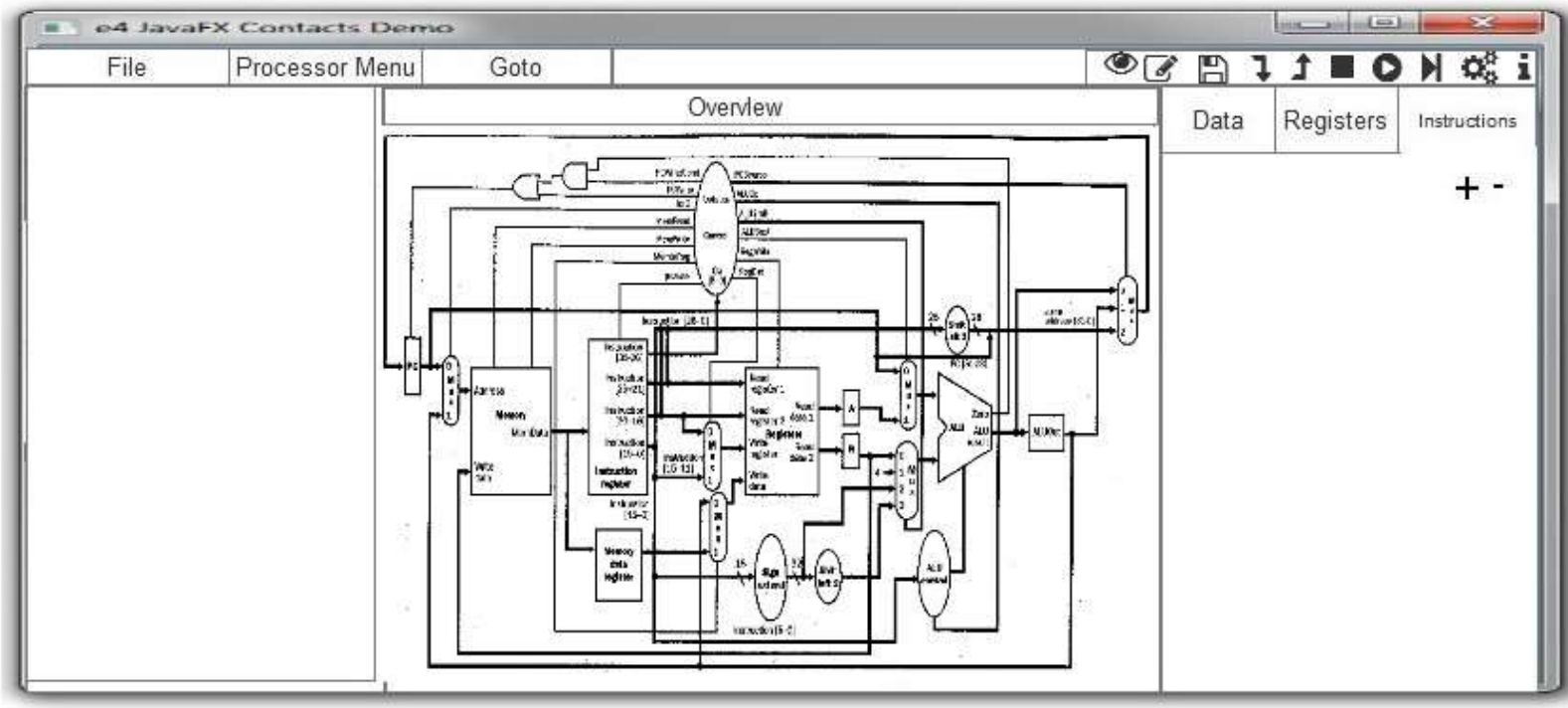
7.3 Use Cases / User Function Description

7.3.1 User Looks/Clicks at/on Component

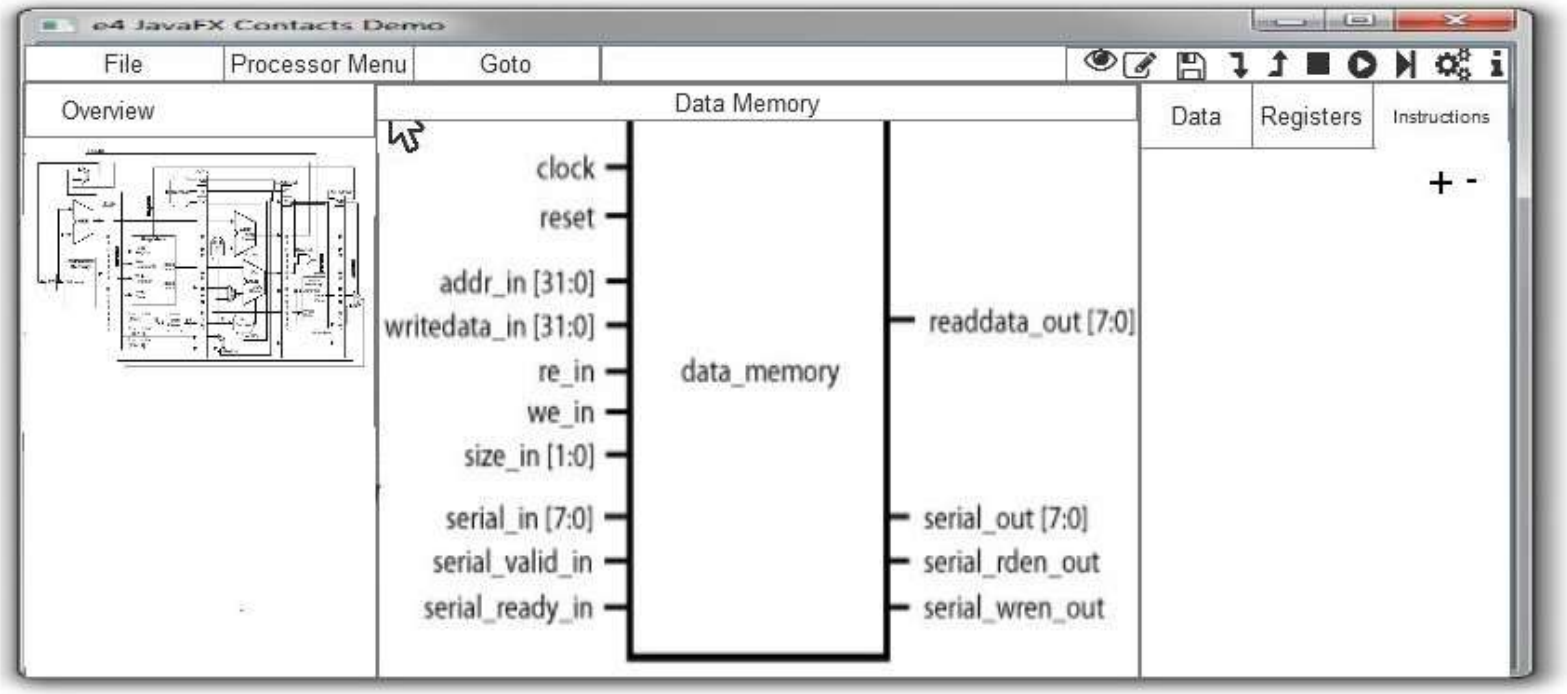
Attributes	Description
Name	Explore a General Component
ID	Explore Component
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to click on the a component to zoom in on the component.
Basic Path	1.The user clicks on the "Component" 2. The system zooms in on the device.
Alternative Paths	
Triggers	The student clicks on the component on the screen/UI Element ().
Preconditions	The student is at a screen that has a component to be explored.
Post-Conditions	None

Overview, user clicks on Data Memory.

1.

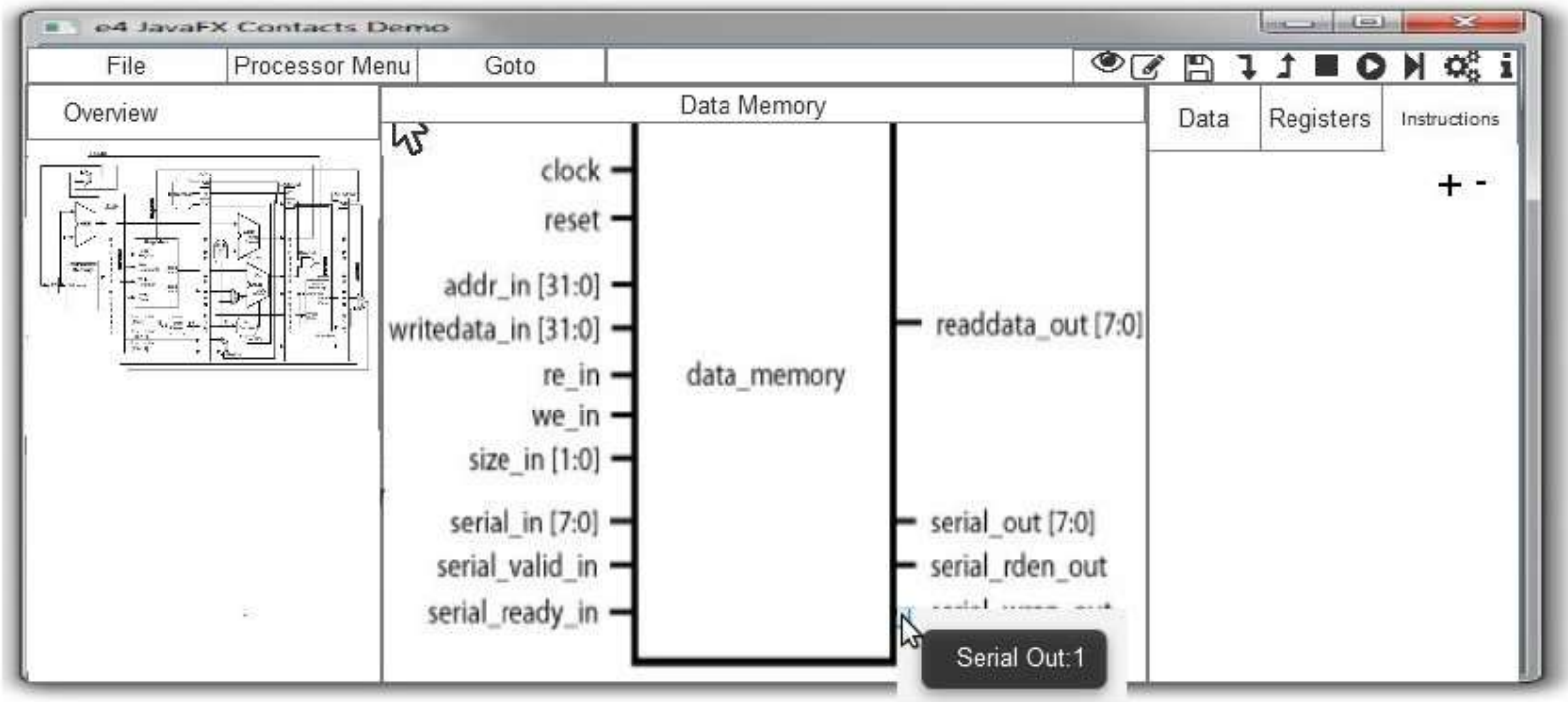


2.


e4 JavaFX Contacts Demo
File
Processor Menu
Goto
Overview
Data Memory
Data
Registers
Instructions
clock
reset
addr_in [31:0]
writedata_in [31:0]
re_in
we_in
size_in [1:0]
serial_in [7:0]
serial_valid_in
serial_ready_in
data_memory
readdata_out [7:0]
serial_out [7:0]
serial_rden_out
serial_wren_out

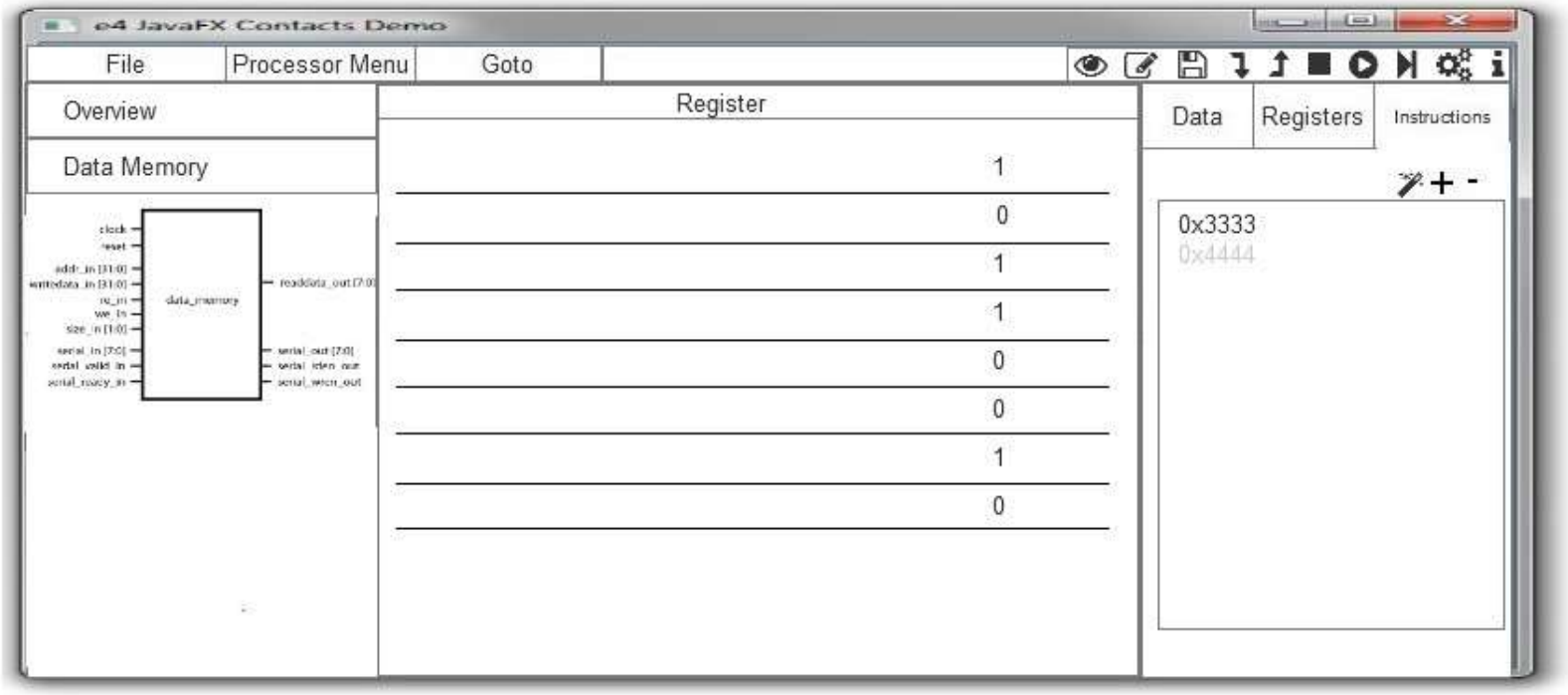
7.3.2 Hovers over device

Attributes	Description
Name	Hovers over device
ID	DEVICE_HOVER
Version	0.1 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to hover over a device within the processor. This will then show a brief summary/description of the device in the left side of the screen.
Basic Path	1.The user hovers on the "Component" 2. The system displays the information on the left side.
Alternative Paths	
Triggers	The student hovers over the component on the screen/UI Element ().
Preconditions	The student is at the overview screen.
Post-Conditions	None



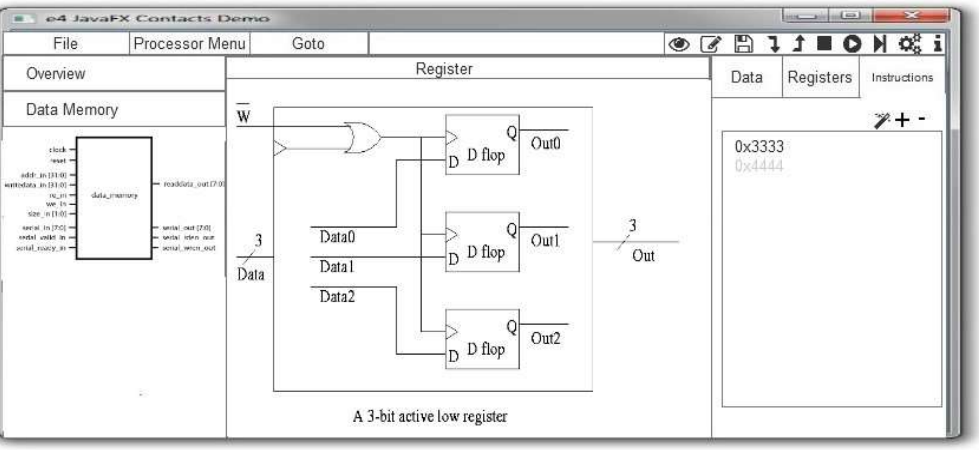
7.3.3 Clicks on Line

Attributes	Description
Name	Student Clicks on a Line
ID	STUDENT_CLICK_LINE
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows an student to click on a line from one unit to another.
Basic Path	1.The user clicks on the "Line" 2.The system displays the line by zooming in on it.
Alternative Paths	At step 2, the user can click back to get back to the previous screen.
Triggers	The student clicks on a line in the screen/UI Element ().
Preconditions	None
Post-Conditions	None



7.3.4 Student Clicks on Eye

Attributes	Description
Name	Student Clicks on a Eye
ID	STUDENT_DISPLAYS_ALL
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows an student to display all the information with the ability to change the data to hex, binary, and abstract.
Basic Path	1.The user clicks on the Eye 2.The system opens up a new popup which displays the data, registers, and instructions data
Alternative Paths	None
Triggers	The student clicks on the eye in the toolbar
Preconditions	None
Post-Conditions	None



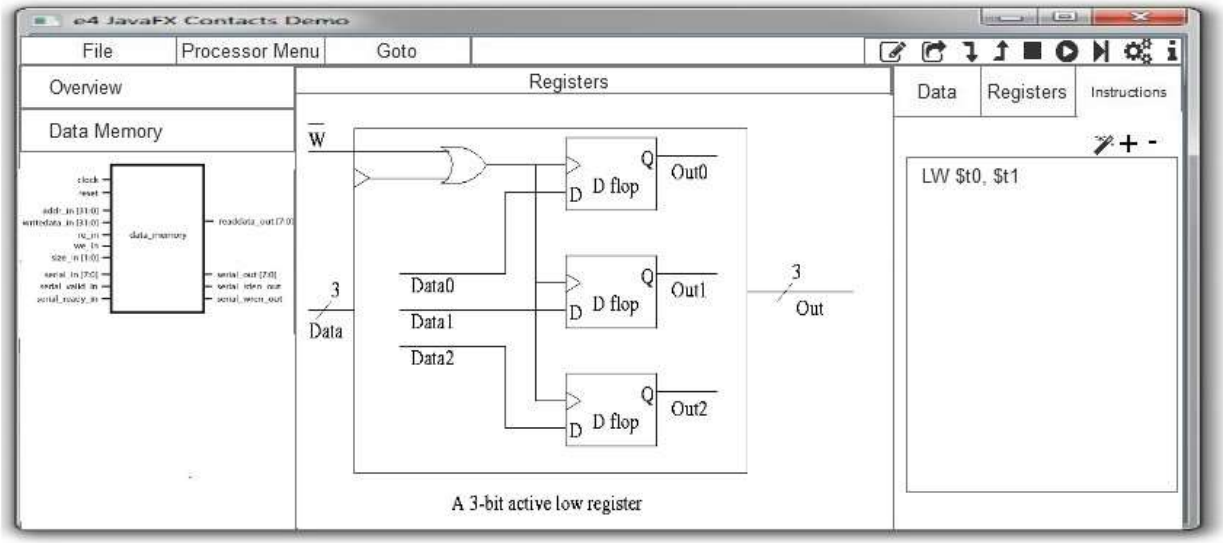
Data	Registers	Instructions
008:0x333	00:0000000	lw($t0,$t1)

7.3.5 Student Clicks on Magic Wand

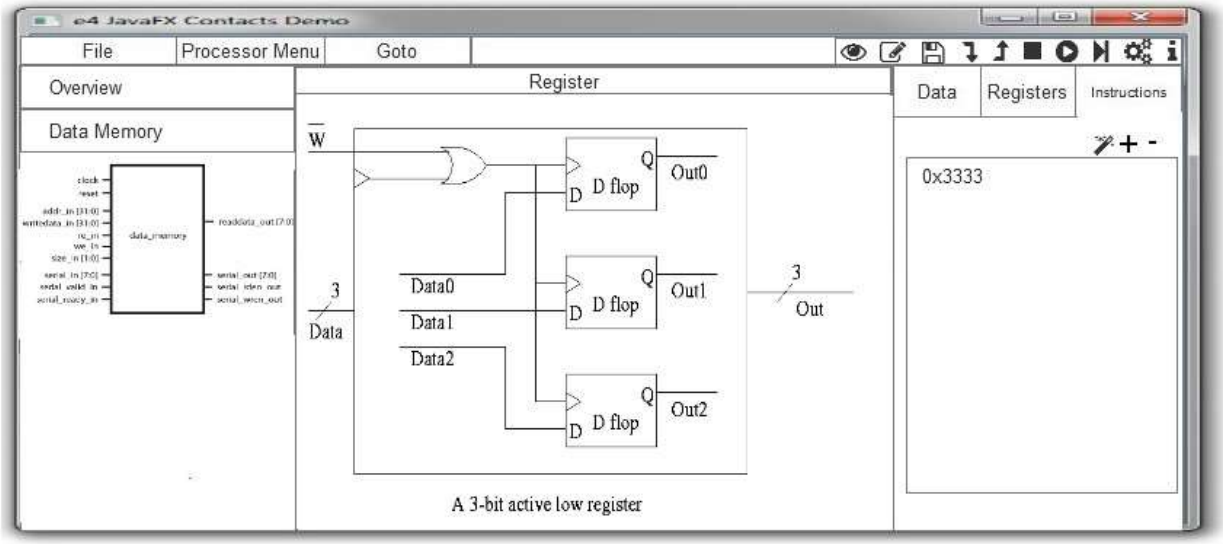
Attributes	Description
Name	Student Clicks on a Magic Wand
ID	STUDENT_CHANGE_DISPLAY
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to have the ability to change the data to hex, binary, and abstract.
Basic Path	1.The user clicks on the Magic Wand 2.The system changes the representation. The data is displayed in hex, binary, decimal, or an assembly instruction (if applicable). It will go in that order every time the user clicks on the wand, and then it will loop back to hex at the end.
Alternative Paths	This button is also available in the data tab, memory tab, and register tab; it is also available in the display all window.
Triggers	The student clicks on the eye in the toolbar
Preconditions	None
Post-Conditions	None

(Pictures on next page)

Before:

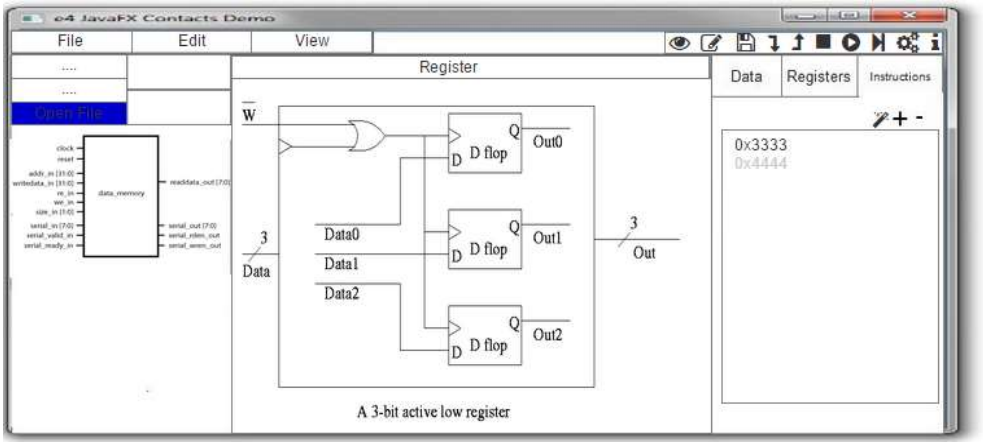


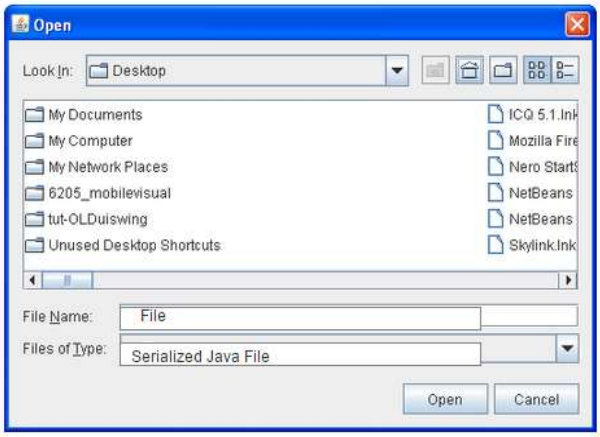
After



7.3.6 Student Clicks on Open File

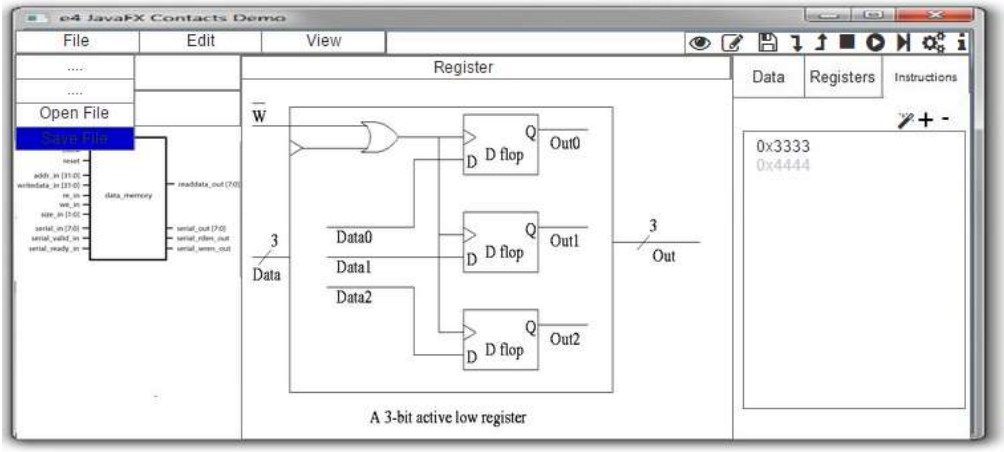
Attributes	Description
Name	Student Clicks on Open File
ID	STUDENT_OPENS_FILE
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to be able to open a processor with preloaded registers, data, and instructions.
Basic Path	1.The user clicks on the file tab. 2.The user clicks on open. 3. The user clicks on the processor that he/she would like to load. This "processor file" would include the instructions, data, register values that were present when the processor was saved.
Alternative Paths	Alt+O will also open the open file window.
Triggers	Alt+O, and the open file in the file menu.
Preconditions	None
Post-Conditions	None

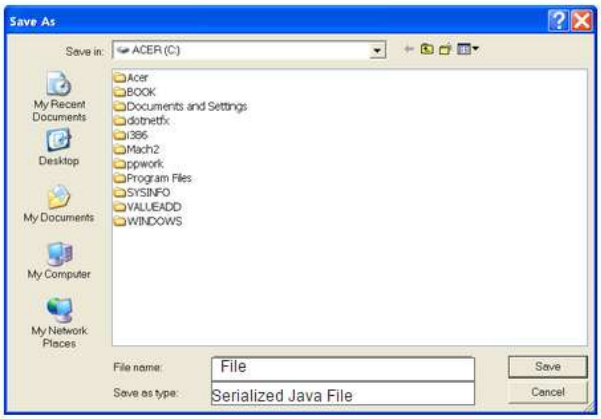




7.3.7 Student Clicks on Save File

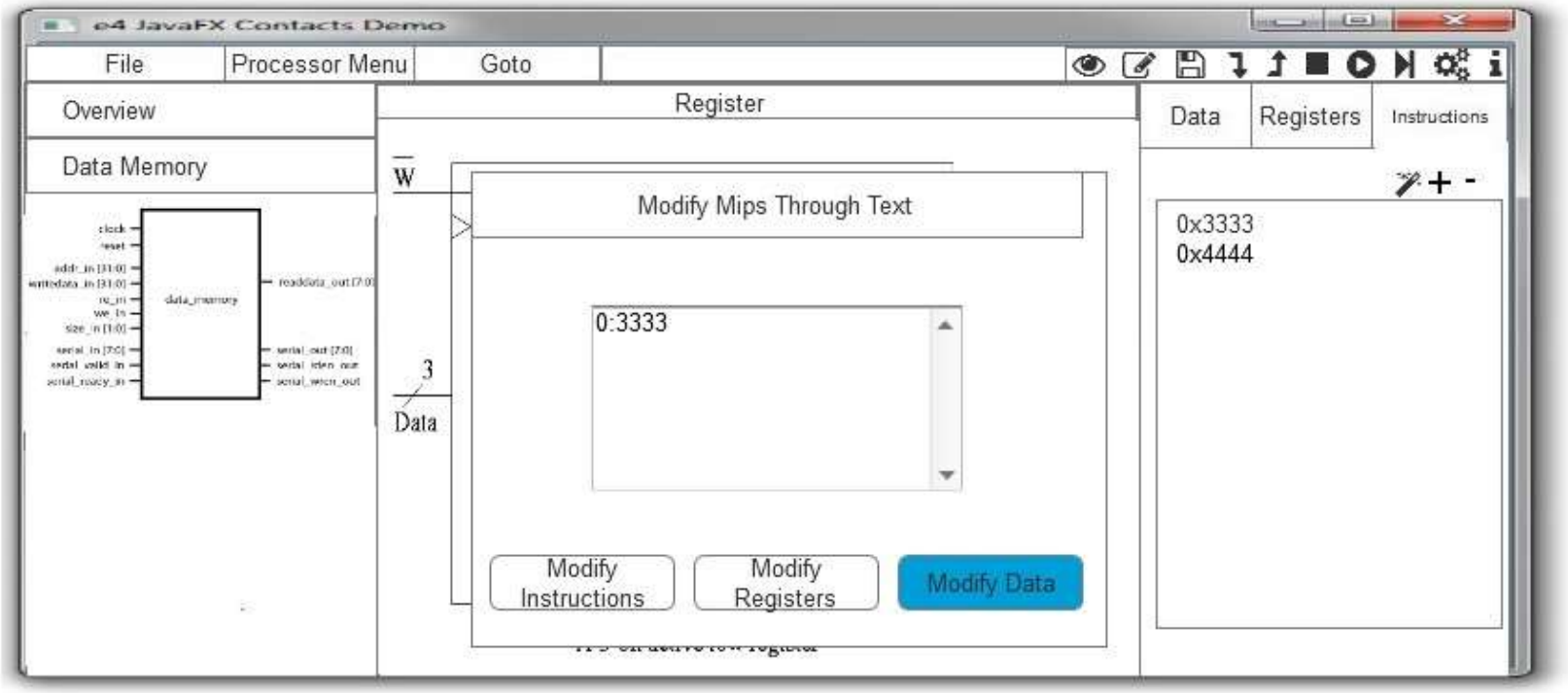
Attributes	Description
Name	Student Clicks Save File
ID	STUDENT_SAVES_FILE
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to be able to save a processor's data, registers, and instructions
Basic Path	1.The user clicks on the file tab. 2.The user clicks on save. 3. The user saves file in location with name as serialized java object.
Alternative Paths	Alt+S will also open the save file window.
Triggers	Alt+S, and the save file in the file menu.
Preconditions	None
Post-Conditions	None





7.3.8 Student Clicks on Edit Icon

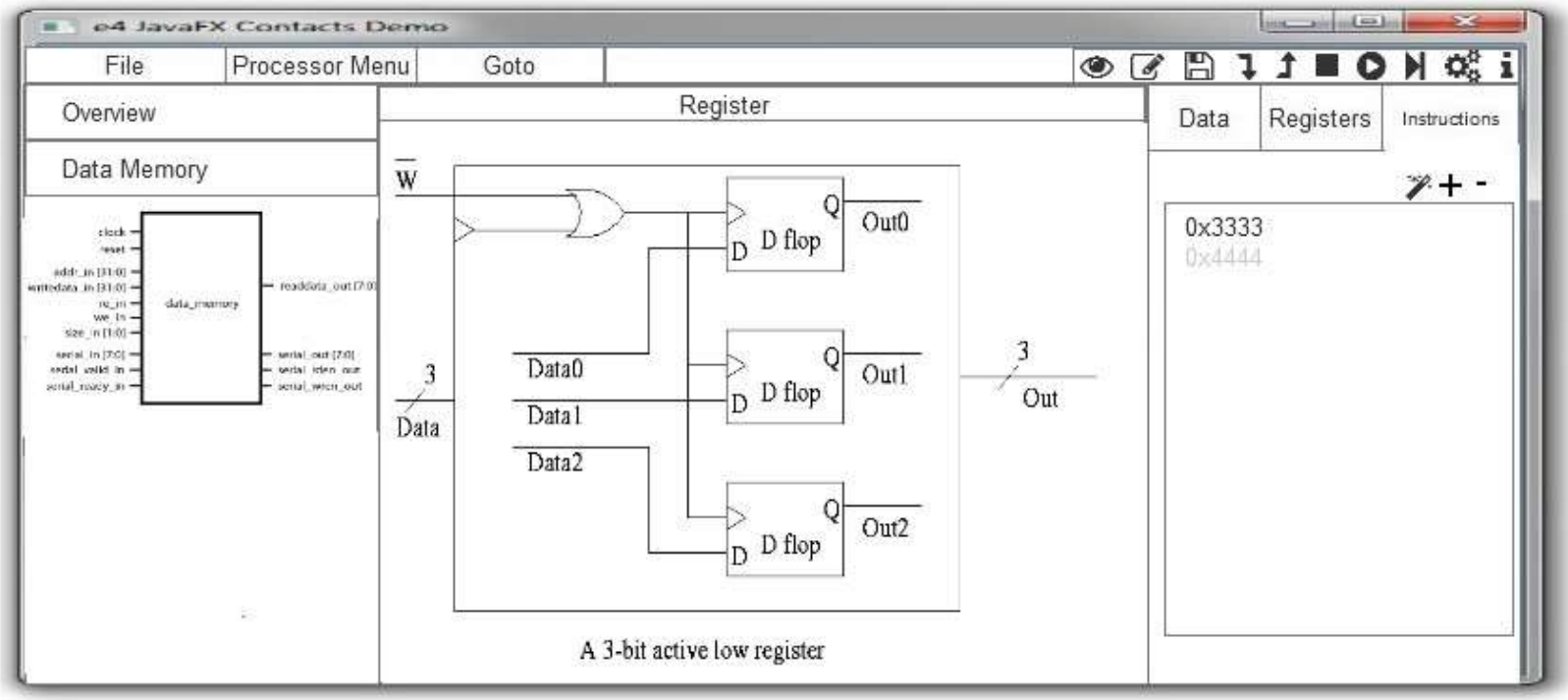
Attributes	Description
Name	Student Clicks The Edit Icon
ID	STUDENT_EDITS_PROCESSOR
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to be able to edit a processor's registers, data, and instruction through text.
Basic Path	1.The user clicks the edit icon (next to the eyeball) in the toolbar.
Alternative Paths	None
Triggers	Edit Button In toolbar
Preconditions	None
Post-Conditions	None



7.3.9 Student Clicks Run Button

Attributes	Description
Name	Student Clicks The Run Button
ID	STUDENT_STARTS_PROCESSOR
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to be able to start the processor.
Basic Path	1.The user clicks the start button.
Alternative Paths	None
Triggers	Play Button in toolbar
Preconditions	The processor must not already be running
Post-Conditions	None

The bold black text is the line that the program is currently running. The subtle grey text has either been run or will be run. It runs subsequently down the list.



7.3.10 Student Clicks Stop Button

Attributes	Description
Name	Student Clicks The Stop button
ID	STUDENT_STOPS_PROCESSOR
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to be able to stop the processor.
Basic Path	1.The user clicks the stop button
Alternative Paths	None
Triggers	Stop Button in toolbar
Preconditions	The processor is being run
Post-Conditions	None

7.3.11 Student clicks Up button

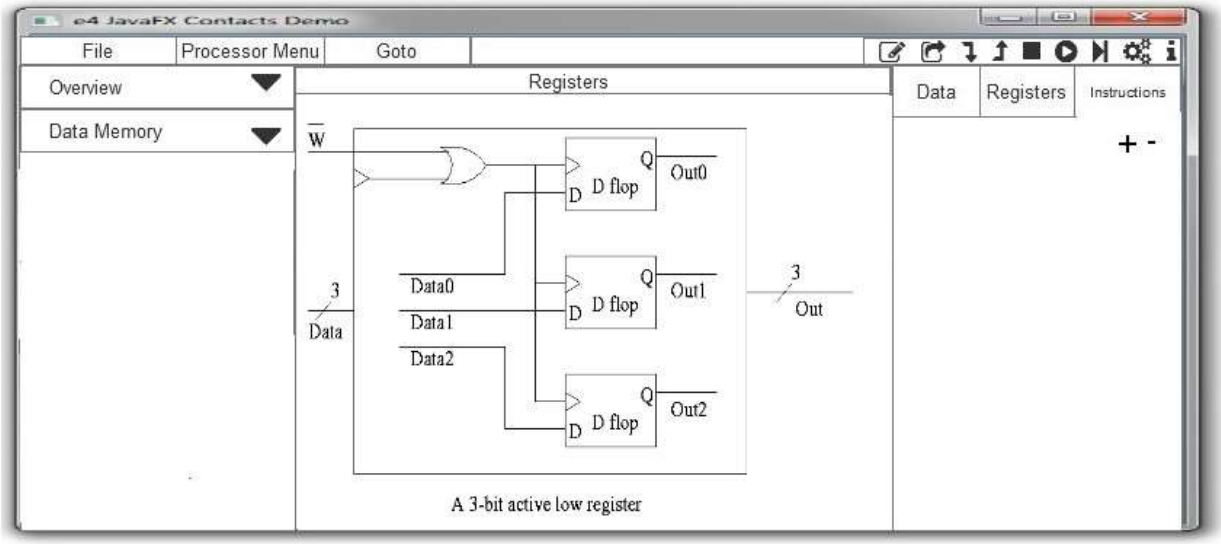
Attributes	Description
Name	Student Clicks The Up button
ID	STUDENT_MOVES_UP
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to be able to move back to the previous view
Basic Path	1.The user clicks the up button
Alternative Paths	None
Triggers	Play Button in toolbar
Preconditions	There is a previous view
Post-Conditions	None

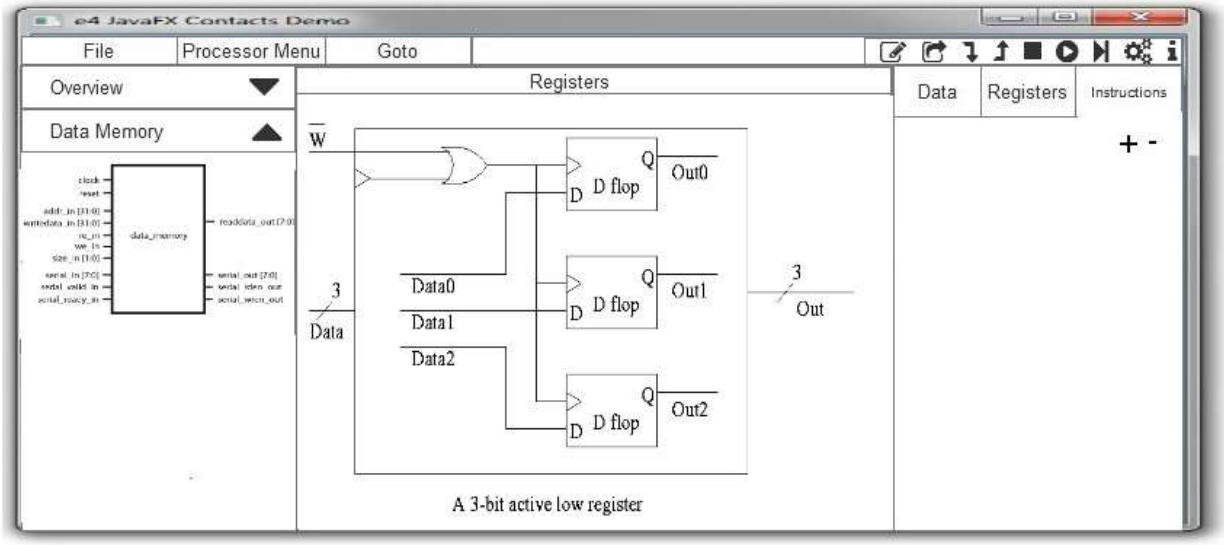
7.3.12 Student Clicks Down Button

Attributes	Description
Name	Student Clicks The down button
ID	STUDENT_GOES_DOWN
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to be able to move down a default step.
Basic Path	1.The user clicks the down button
Alternative Paths	None
Triggers	Down Button in toolbar
Preconditions	Requires a next level to go to
Post-Conditions	None

7.3.13 Student Clicks Tab Caret

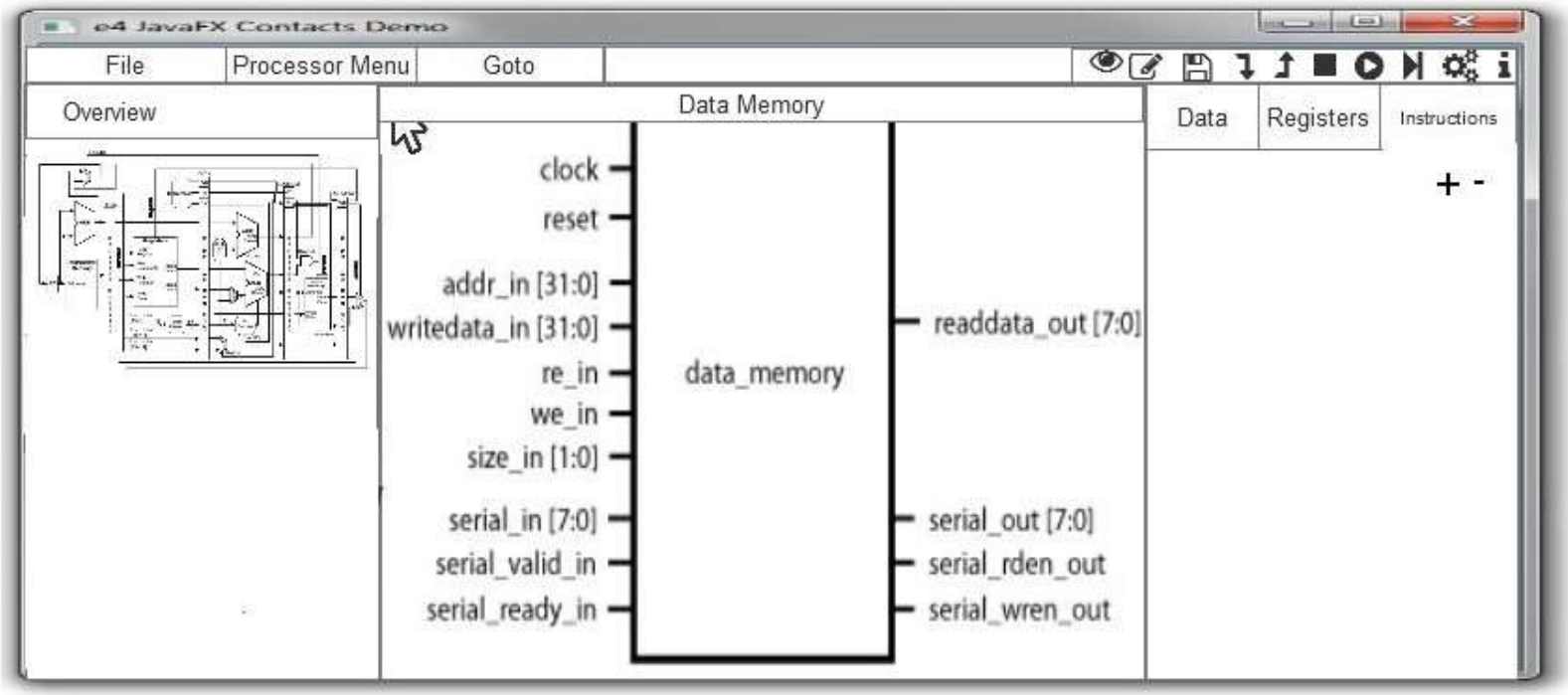
Attributes	Description
Name	Student Clicks Tab Caret
ID	STUDENT_CLICKS_CARET
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to close and open an accordion tab
Basic Path	1.The user clicks the caret button on the left
Alternative Paths	None
Triggers	Caret Button
Preconditions	It requires atleast one previous view
Post-Conditions	None

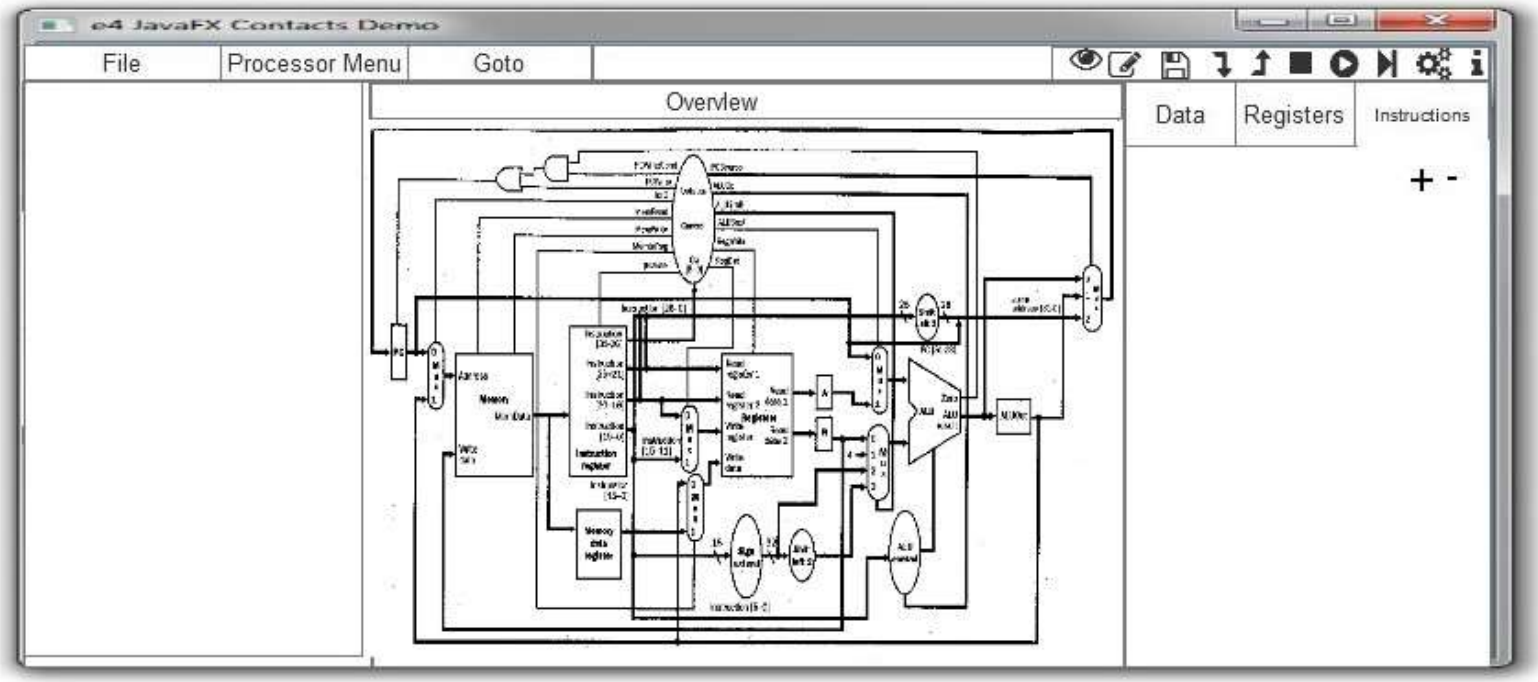



e4 JavaFX Contacts Demo
File
Processor Menu
Goto
Overview
Data Memory
Registers
Data
Registers
Instructions
+ -
W
3
Data
Data0
Data1
Data2
D flop
D flop
D flop
Q
D
Out0
Out1
Out2
3
Out
A 3-bit active low register

7.3.13 Student Clicks on Tab

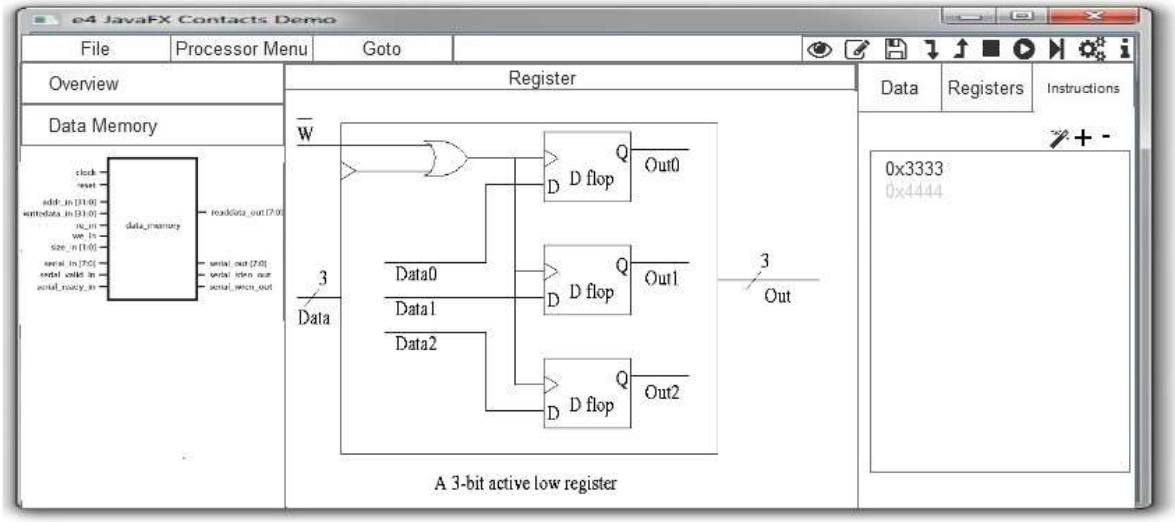
Attributes	Description
Name	Student Clicks On Tab
ID	STUDENT_MOVES_TO_TAB
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to go to a previous view through a tab.
Basic Path	1.The user clicks a tab
Alternative Paths	None
Triggers	Tab click
Preconditions	It requires atleast one previous view
Post-Conditions	None

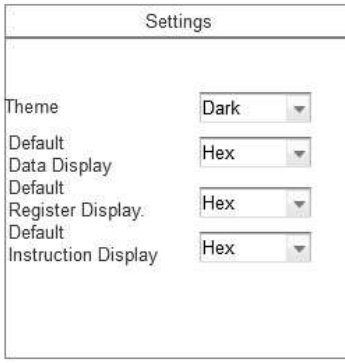



e4 JavaFX Contacts Demo
File
Processor Menu
Goto
Overview
Data
Registers
Instructions
+ -

7.3.14 Student Clicks on Settings Button

Attributes	Description
Name	Student Clicks On Settings Button
ID	STUDENT_CLICKS_SETTING
Version	0.2 (Frank)
Date	26-AUG-2015
Summary	This use case allows a student to go to the settings
Basic Path	1.The user clicks the gears/settings icon in tool bar.
Alternative Paths	None
Triggers	settings button click
Preconditions	None
Post-Conditions	None





Section 8 - Other Interfaces

No external interfaces as of yet.

Section 9 - Extra Design Features / Outstanding Issues

Does not fit anywhere else above, but should be mentioned -- goes here

Section 10 – References

Any documents which would be useful to understand this design document or which were used in drawing up this design.

Section 11 – Glossary

Glossary of terms / acronyms