Sim Machine

Software Requirements Specification

1.00

October 7, 2010

Group 4
Brok Bucholtz
Jose Antony
Mike Easter
Michael Haufe
Ryan Biwer

Revision History

Date	Description	Author	Comments

Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date

Table of Contents

1. Introduction

Our goal is to develop software for the users of class CS151 -Survey of Computer Science at the University of Wisconsin-Milwaukee. The software simulates a simple computer machine. The interface allows users to use simple machine language with a limited list of instructions. The goal of this simulation is to provide students with an understanding of memory allocation and CPU registers using this limited instruction set.

1.1 Purpose

Develop a simple but representative machine language system within specified machine specs for the students and instructors in the course 'Survey of Computer Science'. Reporting of general registers, main memory, program counter and instruction registers are information a user receives throughout the program execution.

1.2 Scope

1. Project SIM is intended to develop a software that simulates a simple machine.
2. The software interface allows the users to enter instructions into memory using a limited list of commands to perform actions. The user will have a graphical user interface to interact with the program, thereby providing a representation of machine language.
3. The main goal of Project SIM is to enable students who may or may not be familiar with computer systems but enrolled in CS151 to develop an understanding of how simple machine languages works with a small machine architecture.
    a. Provide effective reporting of the changes in registers, and memory locations as a result of the command execution.
    b. Generate error reports for unknown or invalid instructions.
    c. Provide a graphical user interface with input lines and instruction options and necessary buttons.

1.3 Definitions, Acronyms, and Abbreviations

- GUI(Graphic User Interface) - The user interface used to interact with the user.
- Machine Cycle (Fetch - Decode - Execute Cycle) - CPU performs its job by continually repeating the algorithm that guides it through this three-step process.
- JSON (JavaScript Object Notation) is a lightweight data-interchange and serialization format.
- General Purpose Register - These registers serve as temporary holding places for data being manipulated by the CPU.
- Special purpose register - These registers include the state of the program including program counter and instruction register.
- PC (Program counter) - PC is a special purpose register containing the address of the next instruction to be executed.
- IR (Instruction register) - IR is used to hold the instruction being executed.
- Data cell - Data cell refers to any general purpose register, main memory cells, and PC.
- Output-Label - GUI text box that is used to display errors to the user.

1.4 References

(1) CPU SIM - Java™ based CPU simulator - cs.colby.edu
http://www.cs.colby.edu/djskrien/CPUSim/cpusimgui.html

(2) JSON Specification- Light weight data interchange and serialization.
http://www.json.org/

(3) Java™ Swing - GUI Toolkit
http://download.oracle.com/javase/1.4.2/docs/api/javax/swing/package-summary.html

1.5 Overview

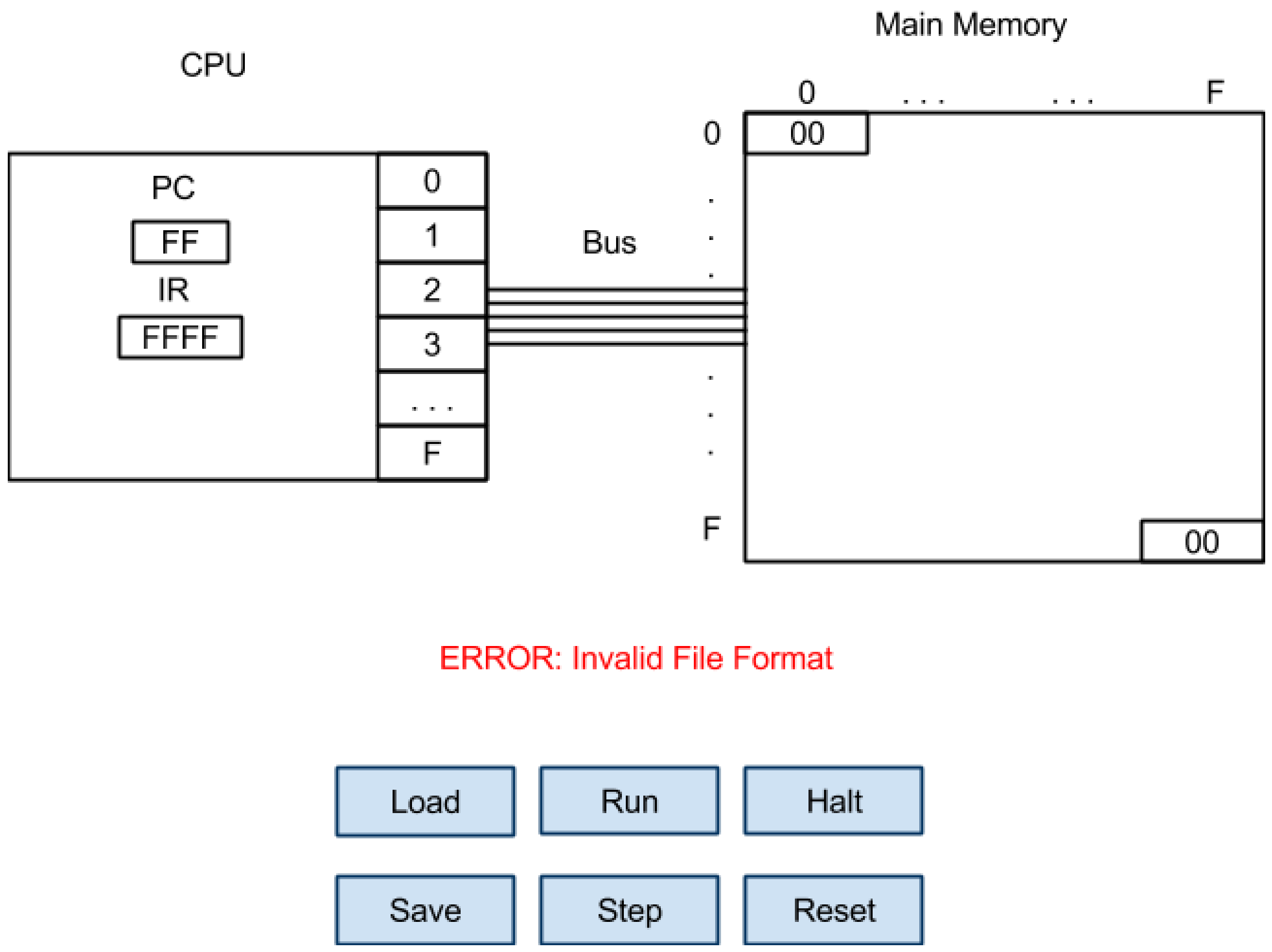
The Project SIM SRS document details specific requirements dealing with input, processing, output and error handling. Use cases, following the specific functional requirements, detail system behaviour for particular cases. Non-functional requirements are also specified followed by UML sequence diagrams to describe loading, saving, halting, resetting, and running the program. Data flow, state-transition, and use case diagrams also analyse the software in specific stages.

2. General Description

Project SIM develops a representative machine language system within specified machine specs for the students and instructors in the course ‘Survey of Computer Science’. The machine has 16 general-purpose registers numbered 0 through F (in hexadecimal). Each register is one byte long with and can be referenced by a unique four-bit value (0000-1111). The machine has 256 bytes of memory. Each memory cell is assigned a unique address from 0 to 255 (00 to FF hexadecimal). Each machine instructions are two bytes long. The first 4 bits provide the op-code and the last 12 states the operand field.

2.1 Product Perspective

Project SIM is to be executed on a platform running Java™. The graphical interface is implemented using the Java™ Swing toolkit [3]. An example of similar product is CPU SIM developed by Colby University, ME [1]. Project SIM intends to represent a micro version of such a project with limited instructions and display information provided below.



2.2 Product Functions

The quick overview of functions involve:

- Loading from register
- Storing the bit patterns in the register
- Moving the patterns from register to register
- Adding bit patterns in register
- Performing logical OR, AND, EXCLUSIVE OR, and ROTATE operation between two registers
- Jumping to an instruction located in memory cell
- Halting the execution process

2.3 User Characteristics

The intended audience of the product includes students enrolled in the 'Survey of Computer Science' course at the University of Wisconsin-Milwaukee. There are no required prerequisites for the class, so the users may or may not have computer/programming experiences before. Therefore it is required that the GUI be standard in organizing menus, labels and buttons according to the logical order of operations.

2.4 General Constraints

The constraints in this program begin with invalid commands. When the program encounters an invalid command, it should alert the user with a message specific to the cause of the problem. The priority of the software is to let the program run as long as it finds a valid op-code. The program is not concerned about validating the fetched information. It is preferred to avoid pop-up windows. Future releases should be able to add features and increase capacities.

2.5 Assumptions and Dependencies

The software is run through Java™ 5.0 Applications Programming Interface. The machine should be capable of running the platform in order for the proper functioning of the software. The program must be able to address the different ways of user inputs and different sequences of order of operation. The information fetched using load operations from the associated file system is not inspected prior to the operation. We are also assuming that the system provides sufficient space to save.

3. Specific Requirements

This section contains details software requirements providing information for design and testing stages. Every input to and output from the system is addressed in the functional requirements.

3.1 External Interface Requirements

3.1.1 User Interfaces

The graphical user interface consists of memory display, register display, input options and execute buttons. The graphical interface is implemented using Java™ Swing toolkit.

3.1.2 Hardware Interfaces

The interactions with the hardware includes saving and loading data from the associated file system.

3.1.3 Software Interfaces

Software interface deals with the actions performed including: load, save, run, step, reset, and halt. The process is implemented through the GUI.

3.1.4 Communications Interfaces

The system will respond to the user for an action performed with a response time less than one second per command.

3.2 Functional Requirements

3.2.1 <Save Program>

3.2.1.1 Introduction

The system shall save the state of the simulation as a JSON formatted file.

3.2.1.2 Inputs

- "Save" Button pressed
- File Name

3.2.1.4 Outputs

- Failure Dialog

3.2.1.4 Processing

The system shall prompt the user with for a location with a popup-dialogue to save the state of the simulation. The PC, IR, general purpose registers will be saved in JSON format at the location entered by the user.

3.2.1.5 Error Handling

3.2.1.5.1 General Error Handling

All errors will display ASCII to output-label, then end this function.

3.2.1.5.2 Specific Error Handling

- If the system fails to save the file, the program will display "ERROR: Failed to Save File" to output-label.

3.2.2 <Load Program>

3.2.2.1 Introduction

The system shall load the state of the simulation from a JSON formatted file.

3.2.2.2 Inputs

- "Load" Button pressed
- File Name

3.2.2.4 Outputs

- Failure Dialog

3.2.2.4 Processing

The system shall prompt the user with a popup-dialogue for the location of a file. The file at the user's location will be loaded. The program will parse using JSON format to get the program state. The state will be loaded into the program.

3.2.2.5 Error Handling

3.2.2.5.1 General Error Handling

All errors will display ASCII to output-label, then end this function.

3.2.2.5.2 Specific Error Handling

- If the system fails to load the file, the program will display "ERROR: Failed to Load File" to output-label.
- If the system fails to load the state from file, the program will display "ERROR: Failed to Load State from File" to output-label.
- If the system fails to parse the file, the program will display "ERROR: Failed to Parse File" to output-label.

3.2.3 <Halt Program>

3.2.3.1 Introduction

The system shall pause the Fetch-Decode-Execute cycle when the 'Halt' button is pressed.

3.2.3.2 Inputs

- "Halt" Button Pressed

3.2.3.3 Processing

The system will stop after the current machine cycle is finished.

3.2.4 <Reset Program State>

3.2.4.1 Introduction

The system shall set all the registers & memory cells to a value of '00' and the PC to 0 if the 'Reset' button is pressed..

3.2.4.2 Inputs

- "Reset" Button Pressed

3.2.4.3 Processing

The system will run 3.2.3<Halt Program> when the 'Reset' button is pressed, then prompt the user for a confirmation with a popup-dialogue that says "Reset Registers and Memory?" If the user clicks "OK", the program will set the all the registers & memory cells to a value of '00' and the PC to 0. If the user clicks "Cancel", the program will continue by executing 3.2.6<Running Program> function.

3.2.5 <Step Though Program>

3.2.5.1 Introduction

The system shall perform 1 machine cycle when the 'Step' button is pressed.

3.2.5.2 Inputs

- "Step" Button Pressed

3.2.5.4 Outputs

- Failure Dialog

3.2.5.4 Processing

The system will run a single machine cycle if the 'Step' button is pressed.

3.2.5.5 Error Handling

3.2.5.5.1 General Error Handling

All errors will display ASCII to output-label, then run 3.2.3<Halt Program> function.

3.2.5.5.2 Specific Error Handling

- If the program fails to fetch, the program will display "ERROR: Failed to Fetch" to output-label.
- If the program fails to decode, the program will display "ERROR: Failed to Decode" to output-label.
- If the program fails to execute the program will display "ERROR: Failed to Execute" to output-label.

3.2.6 <Running Program>

3.2.6.1 Introduction

The system shall run 3.2.5<Step Through Program> after the 'Run' button is pressed.

3.2.6.2 Inputs

- 'Run' Button Pressed

3.2.6.4 Outputs

- Failure Dialog

3.2.6.4 Processing

When the user presses the 'Run' button, the system will start a loop. The program will run 3.2.5<Step Through Program> one after the other until execution is terminated either by the HALT op-code, the user (by pressing the halt button), or an unresolvable error. When an error is committed in 3.2.5<Step Through Program>, this function, 3.2.6<Running Program>, will end.

3.2.6.5 Error Handling

Error handling will be done by 3.2.5<Step Through Program>

3.3 Use Cases

3.3.1 Set Data Cell

ID: 1
Created By: Mike Easter
Date Created: 5/10/10
Last Updated By: Mike Easter
Date Last Updated: 6/10/10

Actors: User
Description: Sets the value of a data cell to a user-specified value. Cells include main memory cells, general purpose registers, the instruction register and the program counter.
Priority: High
Frequency of Use: High
Preconditions:

1. Program is in state where it is waiting for user input.

Postconditions:

1. The data cell's data value is set to the entered value.

Normal Course of Events:

1. User selects a data cell's text field.
2. User enters a value into the data cell's text field.
3. User moves focus away from data cell.
4. Memory cell data contains the entered value.

Alternative Courses:

None

Exceptions:

3a. User enters an invalid value and moves focus away from data cell.

3a1. Error indication is shown, such as by cell printing an error message or by highlighting the data cell.

Includes: None
Special Requirements: None
Assumptions: None
Notes and Issues: None

3.3.2. Execute Instruction

ID: 2
Created By: Mike Easter
Date Created: 5/10/10
Last Updated By: Mike Easters
Date Last Updated: 6/10/10

Actors: User
Description: Executes the next instruction
Priority: High
Frequency of Use: High
Preconditions:

1. User has program open
2. Program is in state where it is waiting for user input

Postconditions:

1. The IR contains the fetched instruction.
2. The PC has been incremented or loaded with a specified memory address by a JUMP s instruction.
3. The cells affected by the instruction's execution are indicated.

Normal Course of Events:

1. User presses "Step" button.
2. Program fetches the instruction from the memory cell addressed by the value in the PC.
3. Program increments the PC.
4. Program decodes the instruction in the IR.
5. Program executes the instruction in the IR and changes the values of the appropriate data cells.
6. Program indicates which data cells were affected by the instruction's execution.

Alternative Courses: None
Exceptions:

- 2a. The value in the PC is not a valid memory cell address.
  - 2a1. Error indication is shown, such as by printing an error message or by highlighting the PC.
- 2b. The value of the instruction register is not a valid instruction.
  - 3b1. Error indication is shown, such as printing an error message or by highlighting the IR and the memory cells the instruction was fetched from.

Includes: None
Special Requirements: None
Assumptions: None
Notes and Issues: None

3.3.3 Run Program

ID: 3
Created By: Mike Easter
Date Created: 5/10/10

Last Updated By: Mike Easter
Date Last Updated: 6/10/10

Actors: User
Description: Fetches, decodes, and executes instructions until program is halted.
Priority: High
Frequency of Use: High
Preconditions:

1. User has program open
2. Program is in state where it is waiting for user input

Postconditions:

1. The instruction register contains the fetched instruction.
2. The cells affected by the program's running are indicated.

Normal Course of Events:

1. User presses "Run" button.
2. Program begins Machine Cycle.
3. Execute Instruction, Steps 2 through 5.
4. Program stops Machine Cycle.
5. Program indicates which cells were affected by the program's instruction.

Alternative Courses:

2a. User presses "Halt" button while program is running.
    2a1. Program finishes executing current instruction, then goes to Step 4.

Exceptions:

2a. The value in the PC is not a valid memory cell address.
    2a1. Error indication is shown, such as by highlighting the PC or by printing an error message.

2b. The value of the instruction register is not a valid instruction.
    2b1. Error indication is shown, such as by highlighting the IR and the memory cells the instruction was fetched from.

Includes:

Execute Instruction Use Case, Steps 2 through 5

Special Requirements: None
Assumptions: None
Notes and Issues: None

3.3.4 Reset Program

ID: 4
Created By: Mike Easter
Date Created: 5/10/10
Last Updated By: Mike Easter
Date Last Updated: 6/10/10

Actors: User
Description: Resets each two consecutive data cells to the HALT command.
Priority: Medium
Frequency of Use: Medium
Preconditions:
1. User has program open.
2. Program is in a state where it is waiting for user input.

Postconditions:
1. Each two consecutive data cells are reset to the HALT command.

Normal Course of Events:
1. User presses “Reset” button.
2. Program confirms with the user that they really want to reset the program.
3. Program resets each two consecutive data cells to hexadecimal digits 00 respectively.

Alternative Courses:

2b. User indicates that they don’t want to reset the program.
    3b1. Program terminates resetting program process.

Exceptions: None
Includes: None
Special Requirements: None
Assumptions: None
Notes and Issues: None

3.3.5 Save State
ID: 5
Created By: Mike Easter
Date Created: 5/10/10
Last Updated By: Mike Easter
Date Last Updated: 6/10/10

Actors: User
Description: Saves the state of the program to a file
Priority: Medium
Frequency of Use: Medium
Preconditions:
1. User has program open
2. Program is in a state where it is waiting for user input

Postconditions:
1. The program’s state is saved to a file with a name specified by the user

Normal Course of Events:
1. User presses “Save” button.
2. Program prompts the user for a name to save the file as, as well as giving them the option to cancel saving.
3. File name does not already exist, and program saves the program’s state to a file.

Alternative Courses:

2a. User indicates that they want to cancel saving the program's state

2a1. Program terminates process of saving state.

2b. File with the entered name already exists

2b1. Program asks the user if they want to overwrite the file

Exceptions:

3b. There is not enough memory to store the file.

2a1. Program indicates there is not enough memory to save and terminates process
of saving program's state.

Includes: None
Special Requirements: None
Assumptions: None
Notes and Issues: None

3.3.6 Open State
ID: 6
Created By: Mike Easter
Date Created: 5/10/10
Last Updated By: Mike Easter
Date Last Updated: 6/10/10

Actors: User
Description: Opens the state of the program from a file
Priority: Medium
Frequency of Use: Medium
Preconditions:

1. User has program open
2. Program is in a state where it is waiting for user input

Postconditions:

1. The values of the data cells are loaded from the file containing the program state.

Normal Course of Events:

1. User presses "Open" button
2. Program prompts the user for the name of the file to be opened and allows them to cancel opening of the file.
3. Program loads the values of all cells from the file and displays their updated values

Alternative Courses:

2a. User indicates that they want to cancel opening program

2a1. Program terminates opening state process.

Exceptions:

2a.. File name entered by user contains invalid characters

2a1. Program indicates invalid file name and returns to Step 2

3a. File does not exist and cannot be opened

3a1. Program indicates that the file does not exist and returns to Step 2

3b. File is formatted incorrectly and cannot be opened

3b1. Program indicates that file is formatted incorrectly and returns to Step 2

Includes: None
Special Requirements: None
Assumptions: None
Notes and Issues: None

3.4 Classes / Objects

3.4.1 Instruction

3.4.1.1 Attributes
An instruction is a machine-language instruction represented as a 4-digit hexadecimal string, with the first digit representing the op-code and the remaining digits representing the operand.

The op-code determines what specific operation the instruction performs. There are 12 different types of instructions, and there are 12 specific op-codes for each. The op-code must be within the range of 1 to C as hexadecimal digits.

The operands determine what memory cells or registers are accessed by the execution of the instruction. The operands are formatted differently for each of the 12 different instructions. Each operand digit must be a valid hexadecimal digit (in the range 0 to F).

3.4.1.2. Functions
An instruction's individual digits can be parsed for decoding and executing the instruction.

3.4.1.3 Use Cases
3.3.2. Execute Instruction

3.4.2 Main Memory

3.4.2.1 Attributes
Main memory is the memory that is directly accessible by the CPU. It contains 256 8-bit memory cells.

3.4.2.2 Functions
Main memory cells can be read from and written to by the processor.

3.4.2.3 Use Case References
3.3.1 Set Data Cell
3.3.2. Execute Instruction
3.3.4 Reset Program
3.3.5 Save State
3.3.6 Open State

3.4.3 Instruction Register (IR)

3.4.3.1 Attributes
The IR is a special purpose register that holds fetched instructions. The IR has a 16-bit capacity, which is the length of an instruction.

3.4.3.2. Functions
Holds the last instruction fetched from main memory for decoding and executing.

3.4.3.3 Use Case References
3.3.2. Execute Instruction
3.3.4 Reset Program

3.4.4 Program Counter (PC)

3.4.4.1 Attributes
The PC is a special purpose register that holds the address of the next instruction to be executed. The PC has an 8-bit capacity.

3.4.4.2 Functions
Holds the address of the first of two consecutive memory cells containing the next instruction to be fetched.

3.4.4.3 Use Case References
3.3.1 Set Data Cell
3.3.2. Execute Instruction
3.3.4 Reset Program
3.3.5 Save State
3.3.6 Open State

3.4.5 General Purpose Register

3.4.5.1 Attributes
A general purpose register is a 8-bit memory cell inside of the processor.

3.4.5.2 Functions
General purpose registers are used to hold values that the processor is currently performing arithmetic or binary operations on.

3.4.5.3 Use Case References
3.3.1 Set Data Cell
3.3.2. Execute Instruction
3.3.4 Reset Program
3.3.5 Save State
3.3.6 Open State

3.5 Non-Functional Requirements

3.5.1 Performance

- Instructions should run at a reasonable rate (< 1 second)

3.5.2 Reliability

- The program should handle invalid input smoothly and reliably.
- If the user enters a non-hexadecimal value, an error should be displayed on the screen.
- If an invalid op-code is encountered during execution of the user's code, execution should stop and an error should be printed on the screen.

3.5.3 Availability

- The program will be available on any computer in the UWM computer labs.

3.5.4 Security

- N/A

3.5.5 Maintainability

- Ability to add extensions to the program may be needed in the future.

3.5.6 Portability

- The user should have the ability to save and load machine states to and from the local file system.

3.6 Design Constraints

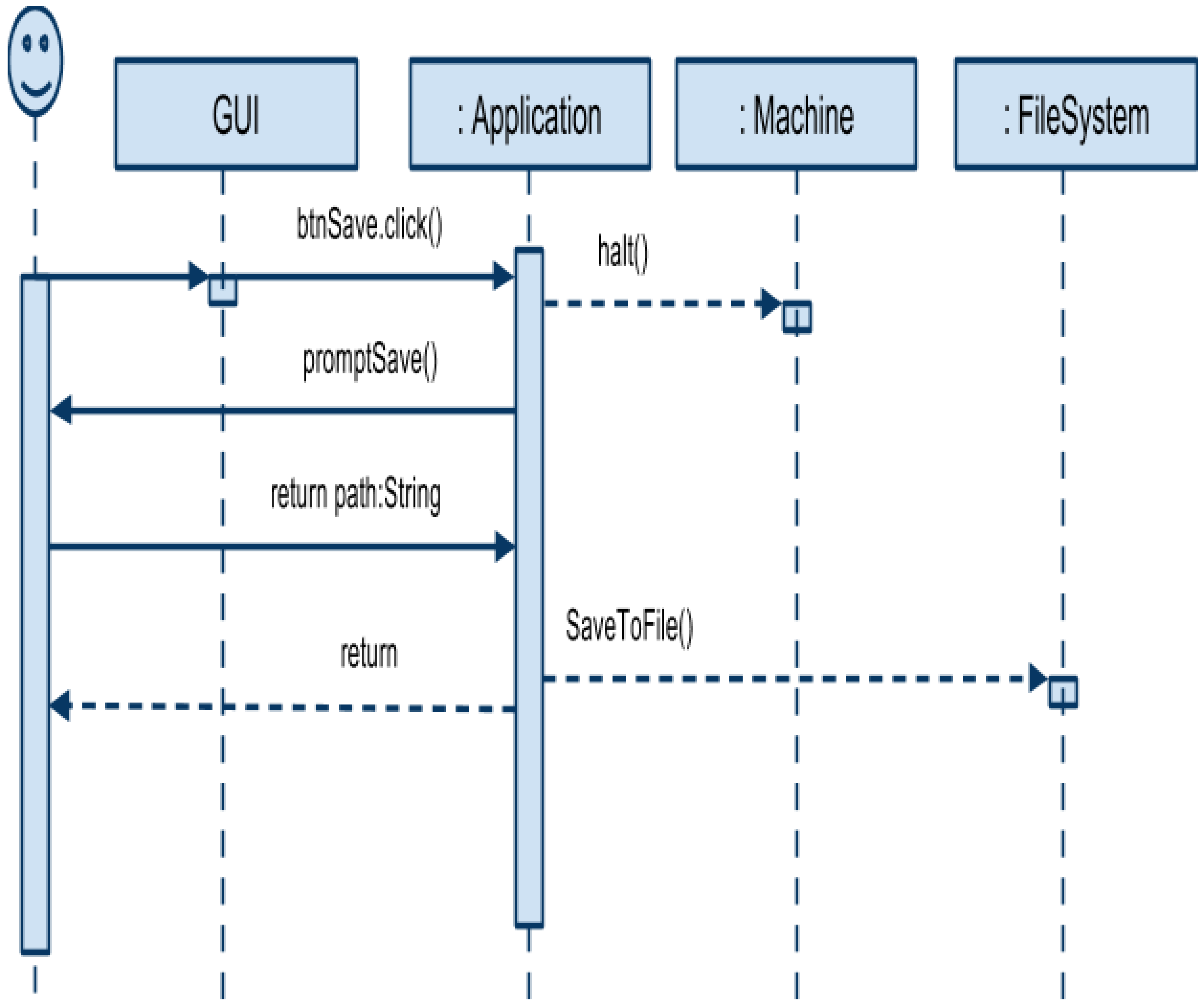
The program should be able to run on any machine running Java™ 5.0+.

4. Analysis Models

4.1 Sequence Diagrams

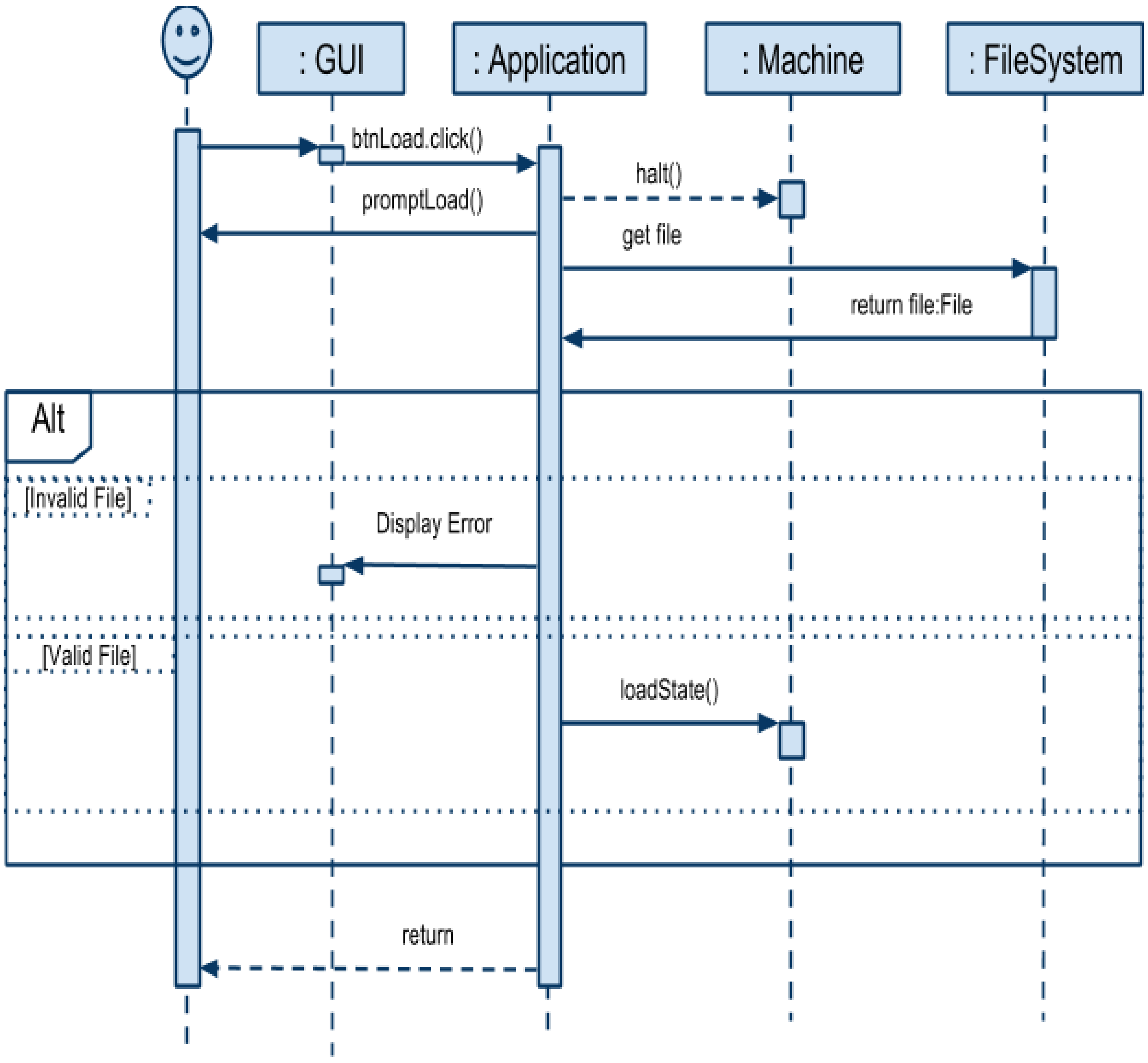
1. **Saving a program**

   When the user clicks on the “Save” button, the currently running user program (if any) is halted (as if the “Halt” button was pressed). The user is then prompted for a save location. The application will then take the current state of the memory locations, PC, IR and general purpose registers and save them to the file specified in JSON format.

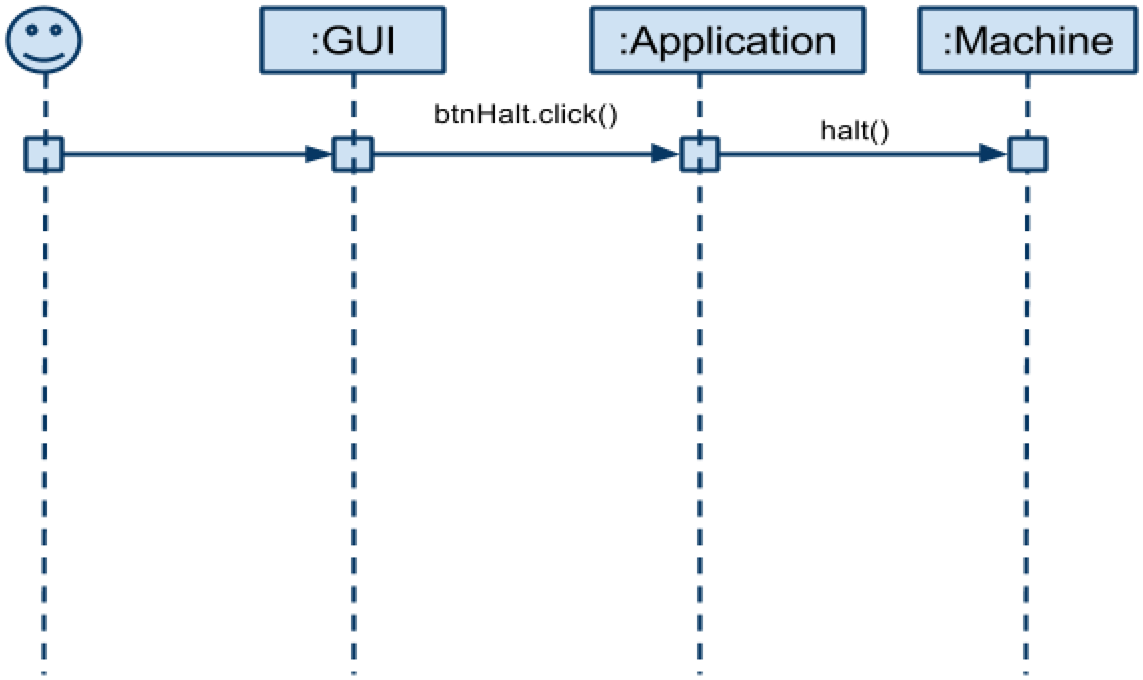


2. **Loading a program**

Loading a program is the inverse of saving it. When the “Load” button is pressed, the currently running user program (if any) is halted (As if the “Halt” button was pressed). The user is then prompted with a common dialog box asking for the file location of a state file (JSON). The respective values will be populated in the machine. If the file selected is invalid, then an error message will be displayed. If the load operation is canceled then no change will occur to the current state of the application

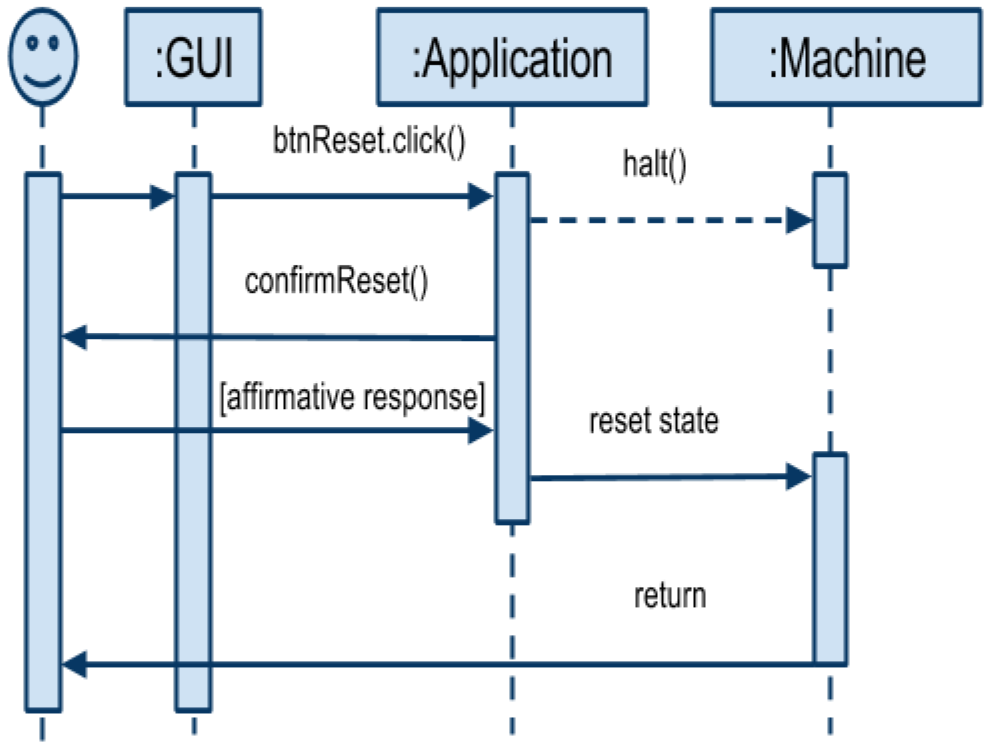


3. **Halting a program**
When the “Halt” button is pressed, the Fetch-Decode-Execute Cycle is halted after the current instruction is executed and the machine state is updated.



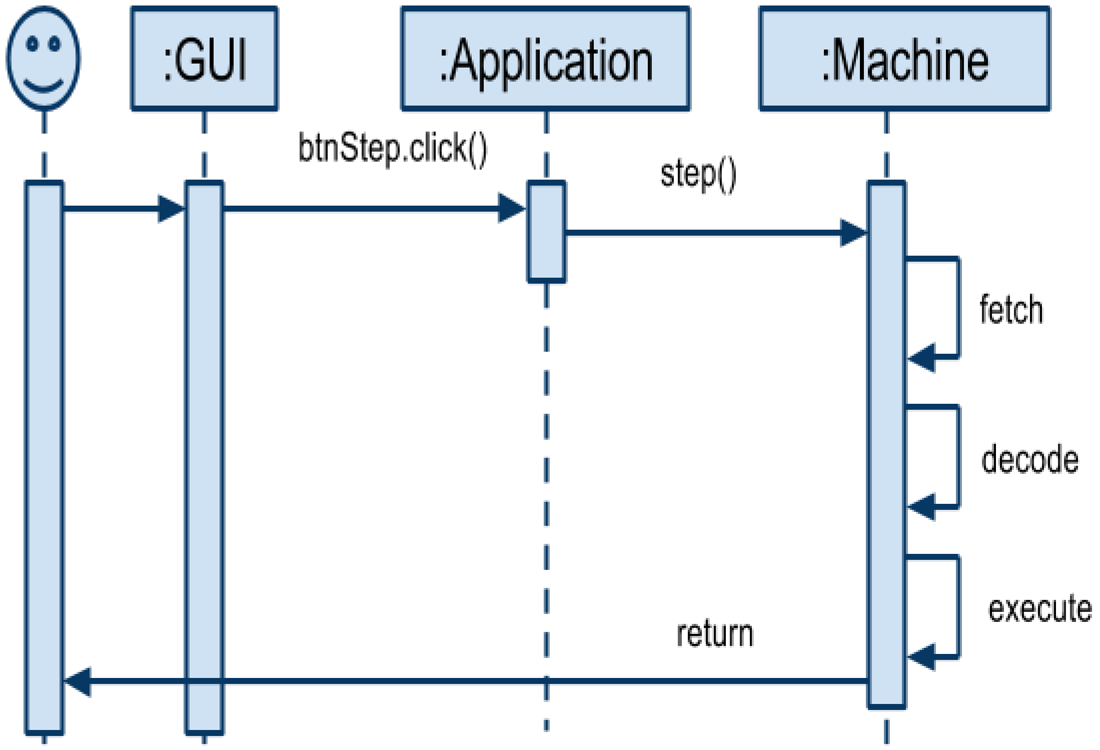
4. **Resetting program state**

When the “Reset” button is pressed, a “Halt” command is issued to the Machine cycle (as if the “Halt” button was pressed). A confirmation prompt is then given to the user.
If the user does not affirm the prompt then no effect will occur.
If the user confirms the prompt, a default value of “00” is placed into every Register and Memory location (Including the IR) , and the PC is reset to 0.



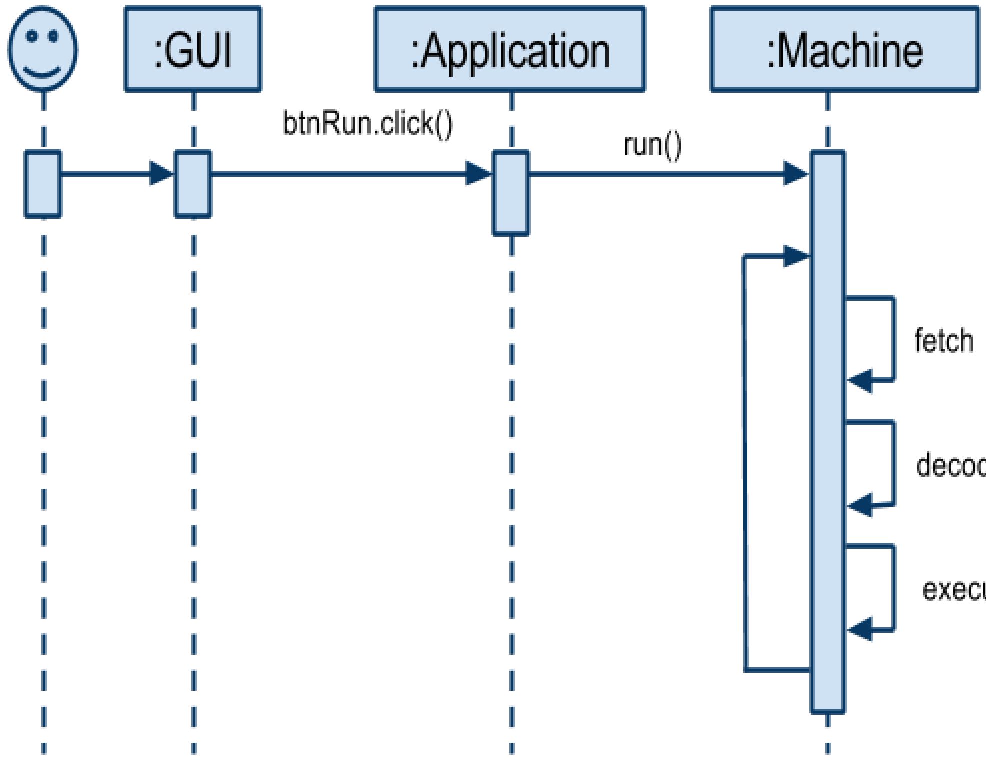
5. **Stepping through a program**

When the "Step" button is pressed, a command is issued to the machine to perform a single Fetch-Decode-Execute cycle.

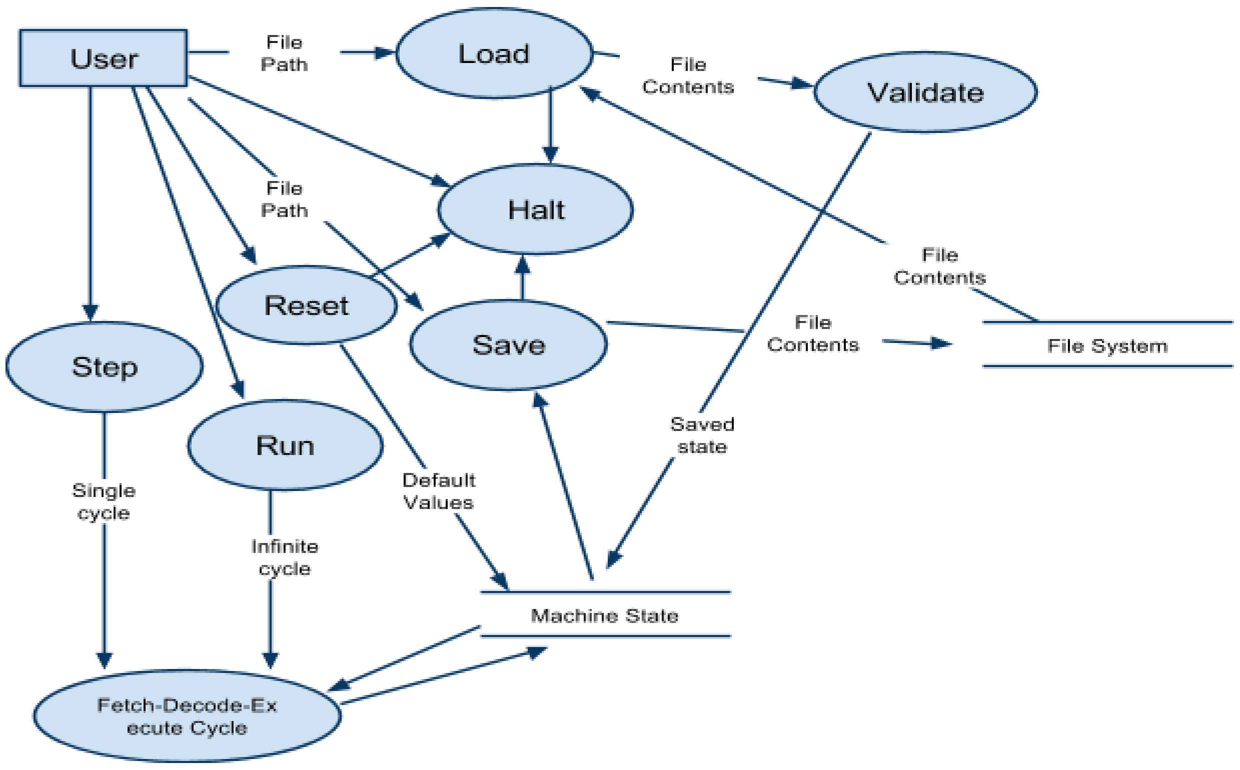


6. **Running a program**

When the “Run” button is pressed, a command is issued to the machine to perform the Fetch-Decode-Execute cycle repeatedly until a halt instruction is encountered, or when the user halts the process by explicitly pressing the “Halt” button, or when the “Halt”command is issued as a side-effect
of pressing the “Save”, “Load”, or “Reset” buttons.



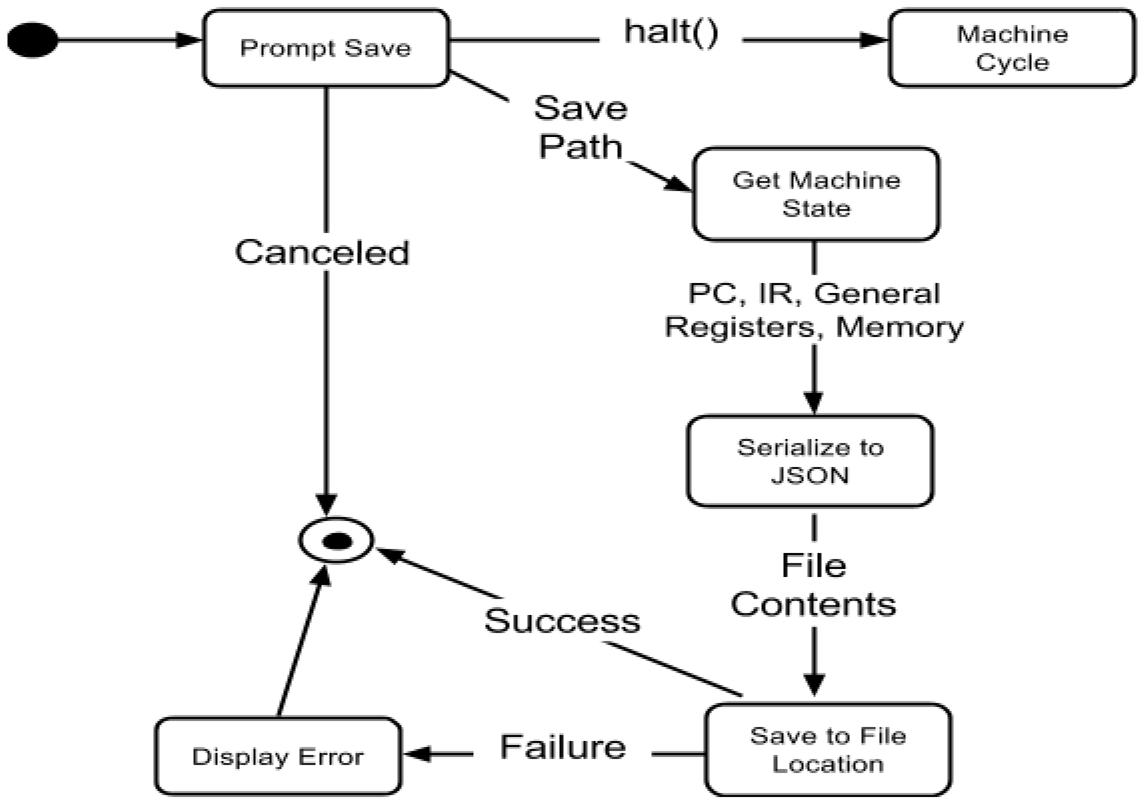
4.2 Data Flow Diagrams (DFD)



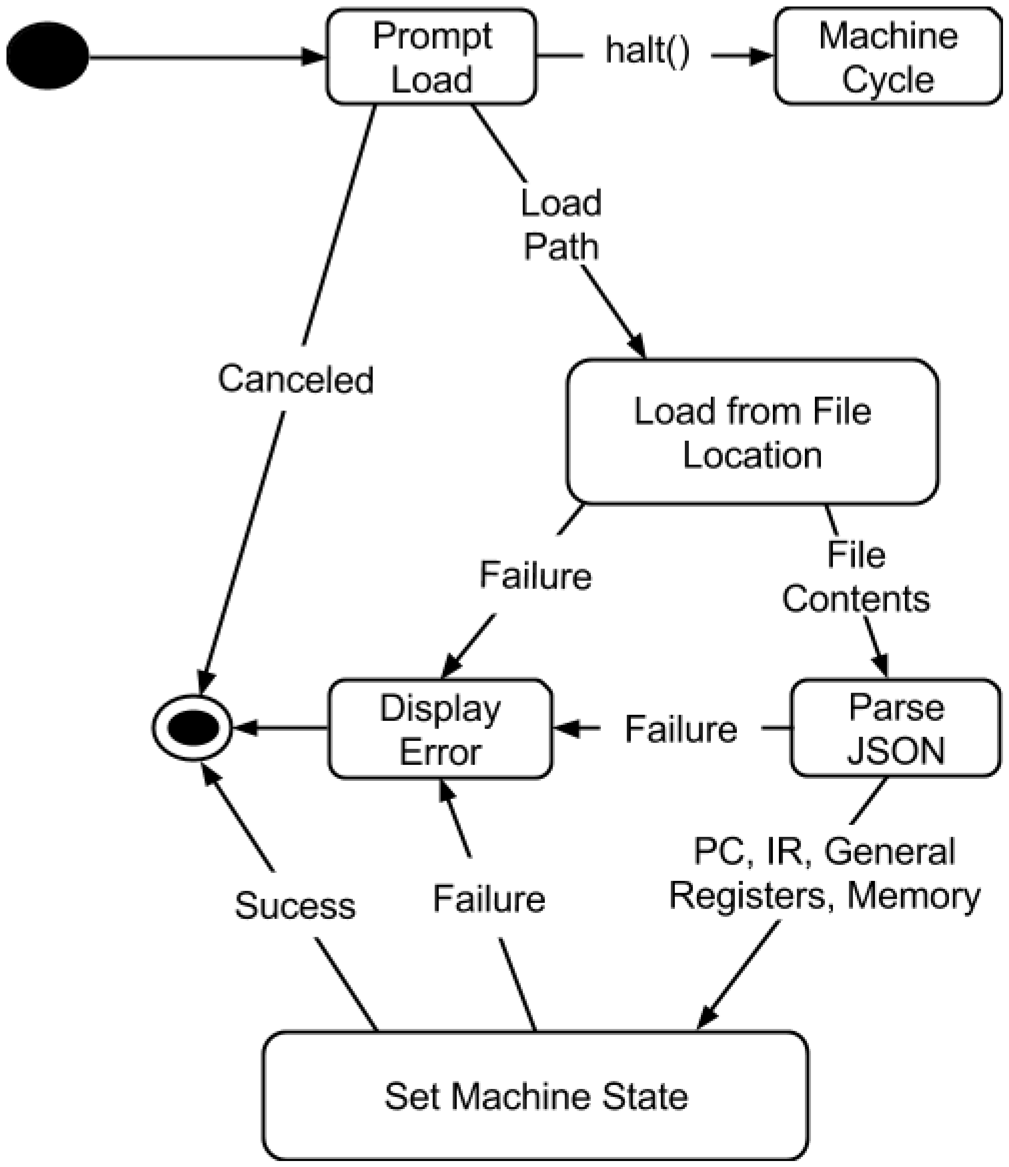
4.3 State-Transition Diagrams (STD)

1. Saving a program

When the user clicks on the "Save" button, the currently running user program (if any) is halted (as if the "Halt" button was pressed). The user is then prompted for a save location. The application will then take the current state of the memory locations, PC, IR and general purpose registers and save them to the file specified in JSON format.

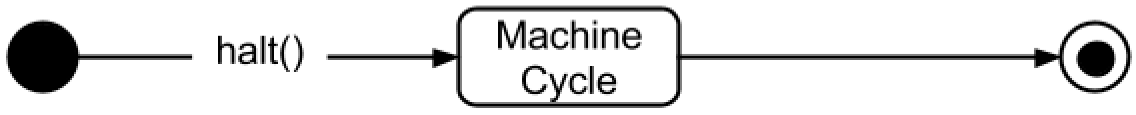


2. **Loading a program**
   Loading a program is the inverse of saving it. When the "Load" button is pressed, the currently running user program (if any) is halted (As if the "Halt" button was pressed). The user is then prompted with a common dialog box asking for the file location of the user's saved program state file (JSON). The respective values will be populated in the machine. If the file selected is invalid, then an error message will be displayed. If the load operation is canceled then no change will occur to the current state of the application.



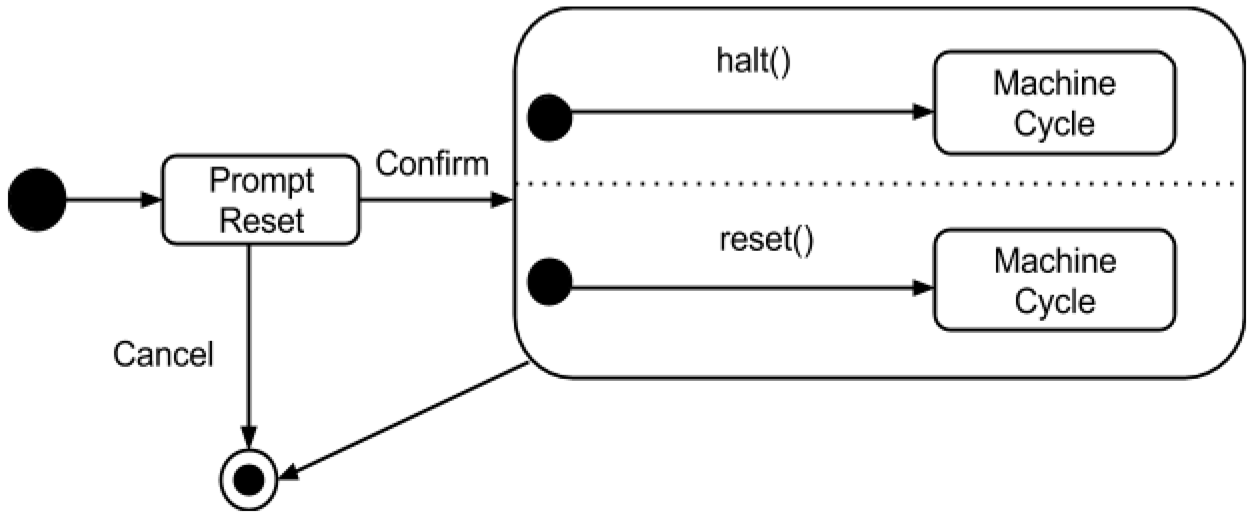
3. **Halting a program**

   When the "Halt" button is pressed, the Fetch-Decode-Execute Cycle is halted after the current instruction is executed and the machine state is updated.



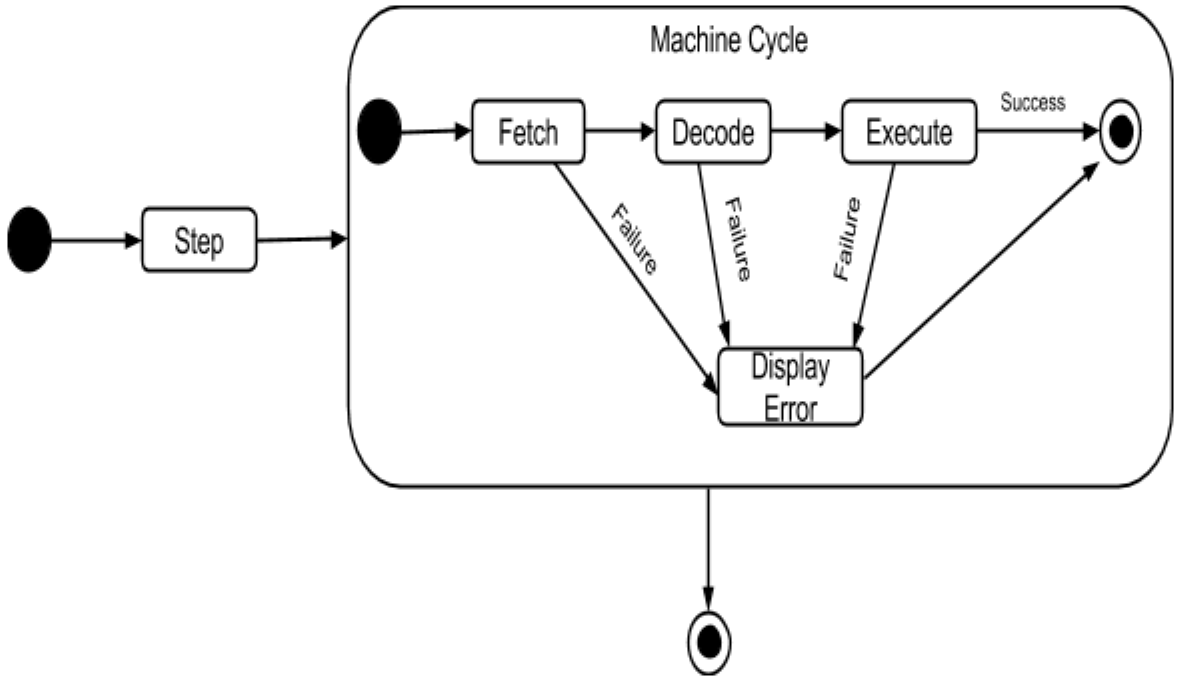
4. **Resetting program state**

   When the "Reset" button is pressed, a "Halt" command is issued to the Machine cycle (as if the "Halt" button was pressed). The HALT op code is placed into every Register and Memory location (Including the IR) , and the PC is reset to 0.



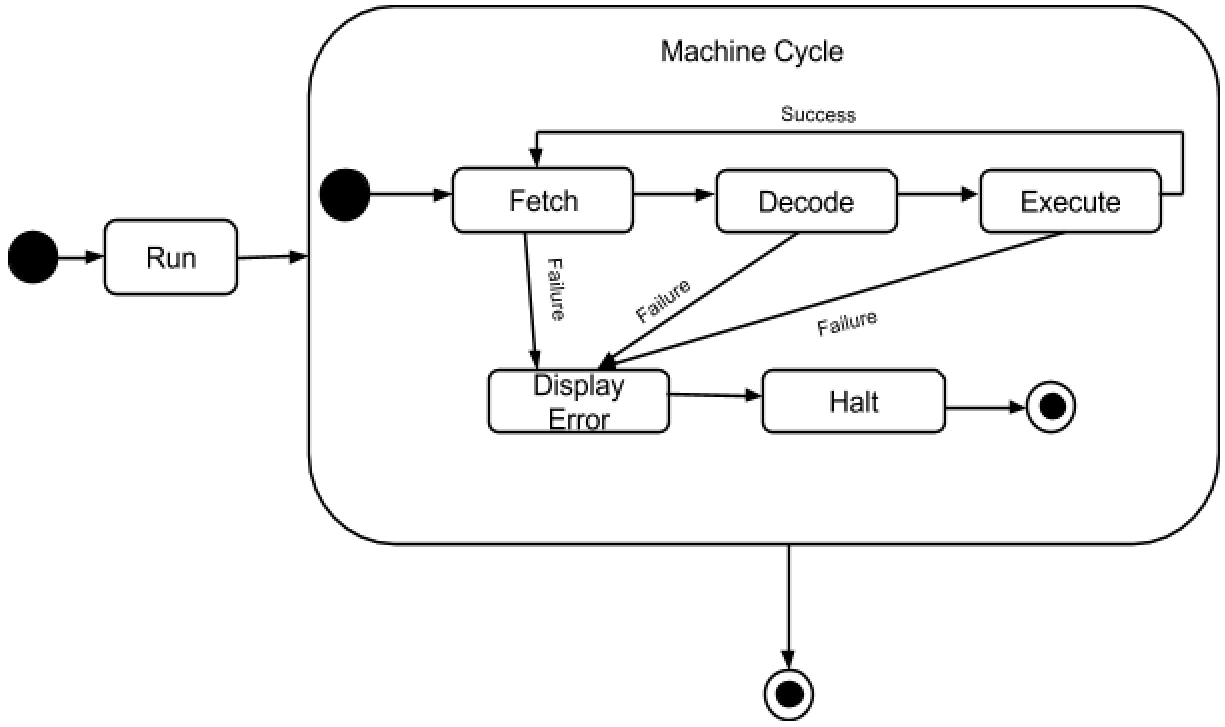
5. **Stepping through a program**

When the “Step” button is pressed, a command is issued to the machine to perform a single Fetch-Decode-Execute cycle.

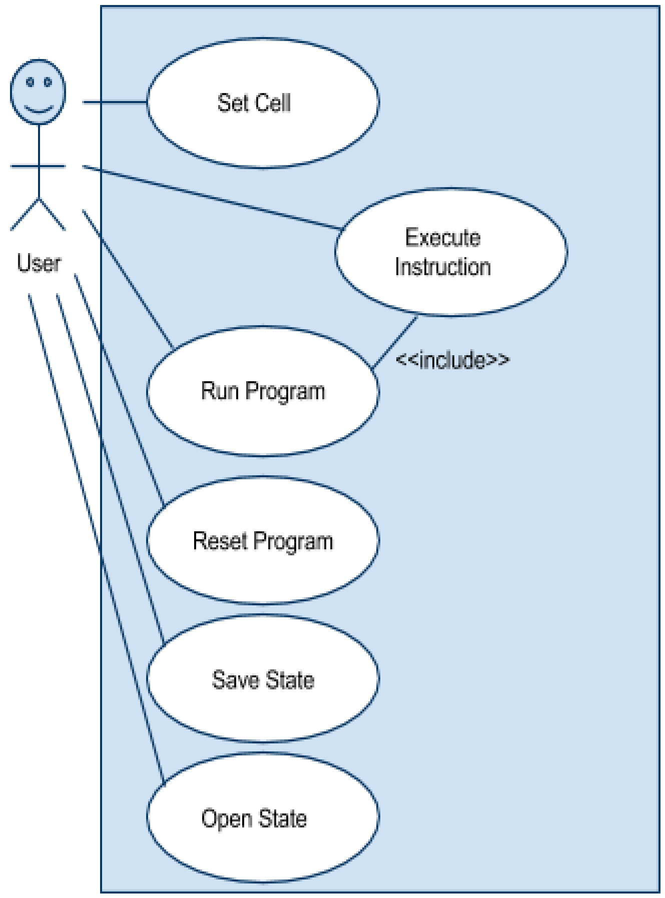


6. **Running a program**

When the “Run” button is pressed, a command is issued to the machine to perform the Fetch-Decode-Execute cycle repeatedly until a halt instruction is encountered, or when the user halts the process by explicitly pressing the “Halt” button, or when the “Halt”command is issued as a side-effect of of pressing the “Save”, “Load”, or “Reset” buttons.



4.4 Use Case Diagram (UCD)



5. Change Management Process

1. The communication process and modification of the SRS document will be conducted through the use of the Google Docs web application.
2. Revision control will be managed by Google Docs for every modifications made by users.

A. Appendices

A.1 Appendix 1

Op-Code	Operand	Description
1	RXY	LOAD the register R with the bit pattern found in the memory cell whose address is XY.
2	RXY	LOAD the register R with the bit pattern XY.
3	RXY	STORE the bit pattern found in register R in the memory cell whose address is XY.
4	ORS	MOVE the bit pattern found in register R to register S.
5	RST	ADD the bit patterns in registers S and T as though they were two's complement representation and leave the result in register R.
6	RST	ADD the bit patterns in register S and T as thought they represented values in floating-point notation and leave the floating-point result in register R.
7	RST	OR the bit patterns in registers S and T and place the result in register R.
8	RST	AND the bit patterns in register S and T and place the result in register R.
9	RST	EXCLUSIVE OR the bit patterns in register S and T and place the result in register R
A	ROX	ROTATES the bit patterns in register R one bit to the right X times. Each time place the bit that started at the low-order end at the high-order end.
B	RXY	JUMP to the instruction located in the memory cell at the address XY if the bit pattern in register R is equal to the bit pattern in register number 0. Other wide, continue the normal sequence of execution.
C	000	HALT execution.