Sim Machine
TEST PLAN
Version 1.00

Group 4

Brok Bucholtz
Jose Antony
Mike Easter
Michael Haufe
Ryan Biwer

Table of Contents

The testing computer must have Java™ and Eclipse™ IDE installed. Before tests can begin, the testing computer must be turned on and booted to the appropriate operating system. The application must be loaded onto the testing computer in an executable form. The source code for the unit test drivers and the classes to be tested must be loaded onto the computer.

1.0 General Information

1.1 Purpose

Conduct functional, integration, and interface testing for project SIM.

1.2 Scope

Test plan covers software requirements and design specification. Plan checks for consistencies with the document and actual software. Software plan also conducts a visual interface testing to ensure smooth running of the software in different platforms.

1.3 System Overview

The main goal of Project SIM is to enable students who may or may not be familiar with computer systems but are enrolled in CS151 to develop an understanding of how simple machine languages works with a small machine architecture. Some of the non-functional requirements to keep in check during testing. All the functional requirements will be tested during unit and integration testing.

- Instructions should run at a reasonable rate (< 1 second)
- The program should handle invalid input with proper messages.
- If the user enters a non-hexadecimal value, an error should be displayed on the screen.
- The program will be available on any computer in the UWM computer labs i.e must be tested on different operating systems used in labs.
- The user should have the ability to save and load machine states to and from the local file system.
- The program should be able to run on any machine running Java™ 5.0+.

Failure to follow any requirements must be dealt according to the software test contingency plan.

1.4 Project References

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

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

1.5 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.

2 Test Definition

2.1 *Function Testing*

Testing each program components. Machine and Instruction class will go through the unit test written.

2.1.1 Requirements to be Tested

- Machine Operation Processing Narratives (Design Document 3.3.3.5.1)

decode()	Takes the instruction from the IR and decodes it into an executable Instruction.
deserialize(file:File)	Reads and sets machine state data from the specified input stream using the specified format.
fetch()	Fetches the contents of the adjacent memory cells addressed by the values PC and PC+1, concatenates them as a 4-bit instruction string,, and stores the result in IR as an integer
getIR() : int	Returns the value of IR.
getMemoryCell(address:int) : int	Returns the value of the memory cell at the specified address.
getPC() : int	Returns the value of PC.
getRegister(address:int) : int	Returns the value of the register at the specified address.
halt()	Halts the machine's execution cycle.
isRunning()	Returns whether or not the machine is in the execution cycle.
run()	Starts the execution cycle
serialize(file:File)	Encodes the machine state to the specified format and writes it to the specified output stream. Throws an exception if

	machine state cannot be written to stream.
setIR(value:int)	Sets IR to the specified value.
setMemoryCell(address:int, value:int)	Sets the memory cell at the specified address to the specified value.
setPC(value:int)	Sets PC to the specified value.
setRegister(address:int, value:int)	Sets the register at the specified address to the specified value.
step()	Fetches, decodes, and executes an instruction.

- Instruction Operation Processing Narratives (Design Document 3.4.3.5.1)

execute()	Behavior determined by subclass implementation: Load 1. Decode register address. 2. Decode memory address. 3. Load value from specified memory address. 4. Store value in specified register. LoadImmediate 1. Decode register address. 2. Decode new value. 3. Store value in specified register. Store 1. Decode register address. 2. Decode memory address. 3. Load value from specified register. 4. Store value in specified memory cell. Move 1. Decode both register addresses. 2. Load value from the first register. 3. Store value in the second register. Add 1. Decode the three register addresses. 2. Load values from second and third specified registers as two's complement numbers.

	3. Add the values together. 4. Store the sum in first specified register. AddFloat 1. Decode the three register addresses.. 2. Load values from second and third specified registers as floating-point numbers. 3. Add the values together. 4. Store the sum in first specified register. Or 1. Decode the three register addresses. 2. Load values from second and third specified registers. 3. OR the values together. 4. Store the result in first specified register. And 1. Decode the three register addresses. 2. Load values from second and third specified registers. 3. AND the values together. 4. Store the result in first specified register. Xor 1. Decode the three register addresses. 2. Load values from second and third specified registers. 3. XOR the values together. 4. Store the result in first specified register. Rotate 1. Decode the register address. 2. Decode number of bits to rotate. 3. Load the value from the specified register. 4. Rotate the value right the specified number of bits. 5. Store the value back in the specified register. Jump 1. Decode the register address. 2. Decode the memory address. 3. Load values from register 0 and the specified register. 4. Compare the two values. 5. If they're unequal, do nothing. 6. Load the value from the specified memory cell. 7. Store the value in the PC. Halt 1. Fetch-Decode-Execute Cycle is stopped

2.1.2 Expected Results

Unit tests for Instruction class and Machine class should produce the expected result for each individual tests. Failure to pass any tests must result in individual check of actual code for any or all classes affected. The error log is displayed and saved from the console of the compiler.

2.1.3 Test Hierarchy

- *Machine Unit Tests (Group 3)*
- *Instruction 1-9 Unit Test (Group 2)*
- *Instruction A-C Unit Test (Group 4)*

2.1.5 Test Data

Test data will be provided by the drivers. If the expected output is different that the actual output the details is added to the error log.

2.1.5.4 Data Recovery

Test results are collected by the drivers and all errors are recorded from the Eclipse console.

2.1.6 Input Commands

- In order to initiate the test of the application, the run button of Eclipse is pressed.
- In order to halt or interrupt the currently running test drivers, the Terminate button is pressed in the Eclipse console.
- The tests are run as a complete set. If an individual test is unsuccessful or incomplete, the entire test set is repeated as a whole vice the individual test case.

2.1.7 Output Notification

All test results and notifications are displayed through the Eclipse console. When the tests are loaded and ready to run, a readiness message is displayed. During the test process, as each individual test succeeds or fails, a message is displayed denoting the details of the test result. Irregularities of input will be managed by the type constraints of the methods being tested. A relevant error message will be displayed in this case.

2.1.8 Support Software

Java Compiler - Eclipse IDE for JAVA

2.1.9 Error Handling

When an error occurs during testing, an error message is displayed on the console denoting the relevant location of the error. When the suspected error is corrected in the program, all tests will be re-run.

2.1.10 Test Conditions

All input will be hard coded as parameters to the respective test cases.

2.2 *Integration Testing*

2.2.1 Requirements to be Tested

- All text fields are editable on the Machine GUI
- Each button performs its role as expected in the Design Document
- Button and editing actions can be performed while a program is in the running state (No GUI lockup)

2.2.2 Expected Results

The expected results are described in the design document for the respective components of the elements being tested.

2.2.3 Test Hierarchy

Machine tests must be successful before GUI and Instruction tests can be performed.

2.2.5 Test Data

The data required for each respective test is limited by the domain described by the type system and invariants specified in the design document. Test data is no applicable for GUI testing as it consists of interface interaction.

2.2.5.1 Test Data Reduction

N/A

2.2.5.2 Input Test Data Control

Where applicable, the minimum, maximum, median and erroneous value will be entered into each unit test as defined by the designated type and invariant constraints.

2.2.5.3 Output Test Data Control

Data produced by testing is displayed on the Eclipse console. Each test displays a pass or fail status and details if required.

2.2.5.4 Data Recovery

N/A

2.2.5.5 Test Data Handling

Tests will be identified by a version identifier in the source code. The maintenance and 3rd party version control of this test data will be performed through D2L

2.2.6 Input Commands

- In order to initiate the test of the application, the run button of Eclipse is pressed.
- In order to halt or interrupt the currently running test drivers, the Terminate button is pressed in the Eclipse console.
- The tests are run as a complete set. If an individual test is unsuccessful or incomplete, the entire test set is repeated as a whole vice the individual test case.

2.2.7 Output Notification

All test results and notifications are displayed through the Eclipse console. When the tests are loaded and ready to run, a readiness message is displayed. During the test process, as each individual test succeeds or fails, a message is displayed denoting the details of the test result. Irregularities of input will be managed by the type constraints of the methods being tested. A relevant error message will be displayed in this case.

2.2.8 Support Software

Java Compiler - Eclipse IDE for JAVA

2.2.9 Error Handling

When an error occurs during testing, an error message is displayed on the console denoting the relevant location of the error. When the suspected error is corrected in the program, all tests will be re-run.

2.2.10 Test Conditions

All input will be hard coded as parameters to the respective test cases.

2.2.11 Test Constraints

N/A

2.3 *Interface Testing*

2.3.1 Requirements to be Tested

- All text fields are editable on the Machine GUI and are constrained on their inputs by the limitations of the underlying datatypes.
- Each button performs its role as specified in the Design Document
- Button and editing actions can be performed while a program is in the running state (No GUI lockup)

2.3.2 Expected Results

- All text fields are editable on the Machine GUI and are constrained on their inputs by the limitations of the underlying datatypes.
- Each button performs its role as specified in the Design Document
- Button and editing actions can be performed while a program is in the running state (No GUI lockup)

2.3.3 Test Hierarchy

Machine tests must be successful before GUI and Instruction tests can be performed.

2.3.5 Test Data

The data required for each respective test is limited by the domain described by the type system and invariants specified in the design document. Test data is no applicable for GUI testing as it consists of interface interaction.

2.3.5.1 Test Data Reduction

N/A

2.3.5.2 Input Test Data Control

Where applicable, the minimum, maximum, median and erroneous value will be entered into each unit test as defined by the designated type and invariant constraints.

2.3.5.3 Output Test Data Control

Data produced by testing is displayed on the Eclipse console. Each test displays a pass or fail status and details if required.

2.3.5.4 Data Recovery

N/A

2.3.5.5 Test Data Handling

Tests will be identified by a version identifier in the source code. The maintenance and 3rd party

version control of this test data will be performed through D2L

2.3.6 Input Commands

- In order to initiate the test of the application, the run button of Eclipse is pressed.
- In order to halt or interrupt the currently running test drivers, the Terminate button is pressed in the Eclipse console.
- The tests are run as a complete set. If an individual test is unsuccessful or incomplete, the entire test set is repeated as a whole vice the individual test case.

2.3.7 Output Notification

All test results and notifications are displayed through the Eclipse console. When the tests are loaded and ready to run, a readiness message is displayed. During the test process, as each individual test succeeds or fails, a message is displayed denoting the details of the test result. Irregularities of input will be managed by the type constraints of the methods being tested. A relevant error message will be displayed in this case.

2.3.8 Support Software

Java Compiler - Eclipse IDE for JAVA

2.3.9 Error Handling

When an error occurs during testing, an error message is displayed on the console denoting the relevant location of the error. When the suspected error is corrected in the program, all tests will be re-run.

2.3.10 Test Conditions

All input will be hard coded as parameters to the respective test cases.

2.3.11 Test Constraints

N/A

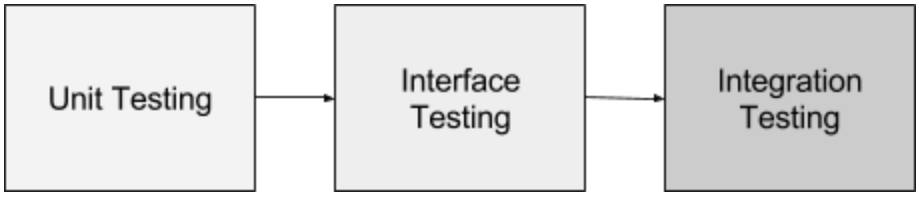
3.0 Test Execution

3.1 Test Schedule

Provide a listing or graphic depicting the locations at which the test will be scheduled and the timeframes during which the test will be conducted. Identify the duration of the test at each site.

Name	Date	Location	Start Time	End Time
Unit Testing				
Instructions	12/15/10	EMS 295	1:00	1:10
Machine	12/15/10	EMS 295	1:10	1:15
Interface Testing				
Opening Application	12/15/10	EMS 295	1:15	1:18
Closing Application	12/15/10	EMS 295	1:18	1:20
Buttons	12/15/10	EMS 295	1:20	1:23
Fields	12/15/10	EMS 295	1:23	1:26
Integration Testing				
Saving Program	12/15/10	EMS 295	1:26	1:30
Loading Program	12/15/10	EMS 295	1:30	1:35
Running Single-Instruction Programs	12/15/10	EMS 295	1:35	1:45
Running Simple Multi-Instruction Programs	12/15/10	EMS 295	1:45	1:55
Running Complex Multi-Instruction Programs	12/15/10	EMS 295	1:55	2:10
Halting Programs	12/15/10	EMS 295	2:10	2:15

3.2 Test Progression



Unit Testing
The first phase is the unit testing of the Machine class and each of the Instruction classes. If any of these tests fail, the appropriate classes must be fixed before proceeding to integration tests, as system cannot work correctly without these essential components.

Interface Testing
The second phase is the testing of the user interface. As the interface cannot be tested without the program being opened, this test along is performed first. Interface testing ensures that each of the specified interface elements are present and that the appropriate fields and buttons react to and accept user input appropriately.

Integration Testing
The last phase of testing is integration testing. Integration testing determines if the application's main function, the running of programs, works properly, with each component and the interface working together to perform this task. Due to the fact that run tests cannot be performed efficiently without loading a sample program, the load and save features are tested first.

3.3 Test Criteria

Test results will be evaluated by each of the testers individually. They will compare the expected results of the tests with the actual results and confirm the presence and function of specified interface components.

3.3.1 Tolerance

Field	Lower Bound	Upper Bound
General-Purpose Register	00	FF
Memory Cell	00	FF
Program Counter	00	FF
Instruction Register	0000	FFFF

3.4 Test Control

Unit testing will be done using written drivers, while interface and integration testing will be done by manual means, with testers verifying that the actual results match the expected results.

3.5 Test Procedures

3.5.1 Setup

The testing computer must have Java™ and Eclipse™ IDE installed. Before tests can begin, the testing computer must be turned on and booted to the appropriate operating system. The application must be loaded onto the testing computer in an executable form. The source code for the unit test drivers and the classes to be tested must be loaded onto the computer.

3.5.2 Initialization

Tests	**Initialization Procedure**
Unit Testing	
Instructions, Machine	Open Eclipse on the testing computer. Import test drivers and classes to test into an Eclipse project.
Interface Testing	
All	Open application on the testing computer.
Integration Testing	
All	Load and open application on the testing computer.
Saving Program	Manipulate data in the PC, IR, general-purpose registers, and main memory to be saved.
Load Program	Create sample programs to be loaded in specified format. Place them in an accessible directory.
Running Programs	Load or manually enter sample programs.
Halting Programs	Run a program.

3.5.3 Preparation

Before even thinking about starting, testers should put their testing caps on. Testers should then request permission to begin testing from Mission Control. Mission control must determine if all systems are go for launch and if weather and environmental conditions are in a suitable state for testing to begin. Testers should set all thrusters to full, and the testing engines should be activated.

3.5.4 Termination

A test is terminated if each of the testers both verifies the results of the test correctly three consecutive times. If a test fails once, then it must be verified as passing three times again after it is fixed, even if it did not fail before. Three is a magic number.

3.5.5 Test Cycle Performance Activities

#	Step	Description
1	Run Test	Build, run, and compile the Java project, using the appropriate test driver as the main class.
2	Verify Results	Manually compare actual results with expect results, noting any errors in the output.
3	Correct Errors	Fix the errors in the code.
4	Rerun Test	Rerun the test, regardless of if the last run was error free.