WASHINGTON STATE UNIVERSITY
VANCOUVER

Software Requirements Specification

for

<MOS 6502 Virtual Machine>

Version <Alpha 0.01>

Prepared by

Group Name: ***<GIT er done>***

Shanpreet Singh	**011586218**	**Shanpreet.singh@wsu.edu**
Aleksandr Khavarovskiy	**011600326**	**a.khavarovskiy@wsu.edu**
Daniil Kernazhytski	**011144879**	**daniil.kernazhytski@wsu.ed**
Saeed Al-Sarhi	**011591809**	**saeed.alsarhi@wsu.edu**
Josh Feener	**011579239**	**joshua.feener@wsu.edu**

Date: **October 15th, 2017**

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
Draft Type and Number	Full Name	Information about the revision. This table does not need to be filled in whenever a document is touched, only when the version is being upgraded.	00/00/00

1 Introduction

The following document is a Software Requirements Specification (SRS). You will find a detailed description of the overall scope of how we intend to build our software using MOS 6502 microprocessor to emulate Nintendo Entertainment systems.

1.1 Document Purpose

The purpose of this document is to identify the product whose software requirements are specified in this document which is a Virtual Machine emulating a MOS 6502 microprocessor. The document will also describe the scope of the product.

1.2 Product Scope

The software being developed will recreate the instructions of the MOS 6502 microprocessor in a modern programming language. In efforts to recreate the classic technologies such as the APPLE 1, APPLE 2, Commodore 64, and the Nintendo Entertainment system (NES) and the Super Nintendo Entertainment system (SNES). The benefits of such an exercise are mostly educational to deeper understand the technologies that came before us.

1.3 Intended Audience and Document Overview

This document is intended for those who are interested in the inner workings of the MOS 6502 emulator, as well as some of the challenges presented in the implementation. In further revisions of this document you can expect to find detailed system requirements and documentation regarding the use of the software. There will also be a list of features and supported emulated systems. Licensing issues will also be addressed in future revisions of the document.

1.4 Definitions, Acronyms and Abbreviations

CSG - Commodore Semiconductor Group
MOS - Metal Oxide Semiconductor.
NES - Nintendo Entertainment System
SNES - Super Nintendo Entertainment System

1.5 Document Conventions

Such fonts or highlighting is to exemplify the importance and significance. The formatting of this SRS is to divide and build space between parts of this document into pieces for the reader to view without any conflicts.

1.6 References and Acknowledgments

IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

2 Overall Description

2.1 Product Perspective

The MOS 6502 Emulator is a self-contained piece of software that is designed to execute external code that normally only runs on hardware that contain the 6502 instruction set. The emulator simulates the functionality of MOS 6502 instructions and acts as a bridge between the 6502 instruction set and the native CPU of the system. The architecture can vary based on the compiler used.

2.2 Product Functionality

The system should accurately execute MOS 6502 code for any chosen emulated system. A user should be able to load and use any software executable by the 6502 Emulator, such as

- Apple 1
- Apple 2
- Commodore 64
- Nintendo Entertainment System

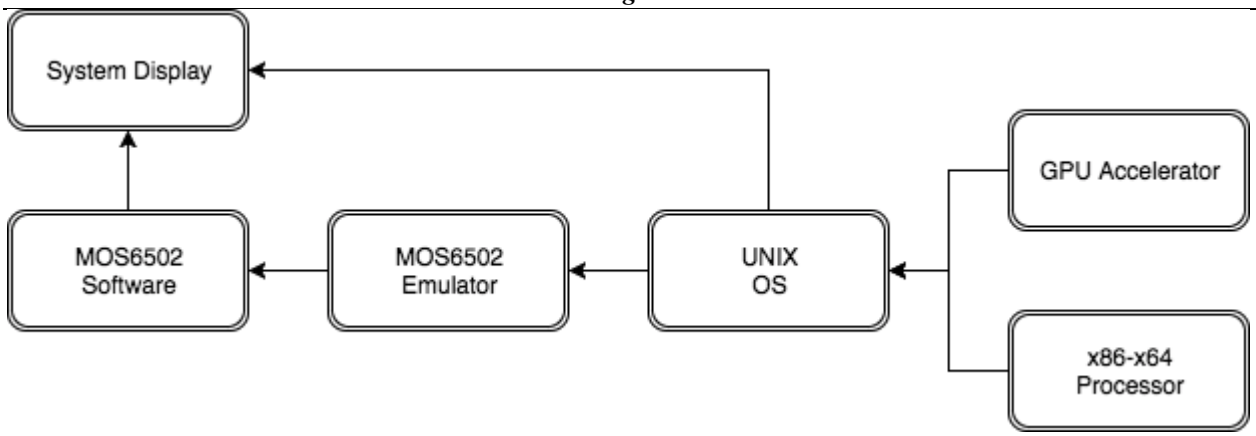
The system should retain all the functionality as intended by the publishers of MOS 6502 software.

2.3 Users and Characteristics

We anticipate users of all technical levels who would like to use our emulator to run 6502 software for historic or nostalgic purposes. Our goal is to provide an authentic experience for users who have native knowledge of the emulated software and users attempting to run the software for the first time.

2.4 Operating Environment

The software will be initially designed to run on a UNIX platform with tentative plans to extend our software to a Windows platform. Our software will be compiled for x64 and x86 systems and tested with Linux and Mac systems. The system must have a graphical processor which should allow rendering via the openGL rendering system. Minimum system requirements will be updated as the development of the software progresses.



2.5 Design and Implementation Constraints

The MOS 6502 does not have a real-time clock and the application timing is defined by the internal clock. This means that the program execution must be constrained by the selected clock speed. Furthermore, certain instructions have conditional execution cycles which must be accurately accounted for to achieve a synced and smooth application execution. The execution time of our application will vary based on the CPU of the system it is running on. This difference of execution timing must be accounted for as well. Minimum system hardware requirements will be updated as the development of the software progresses.

2.6 User Documentation

The documentation will be developed alongside the software and will include:

- Installation of the software
- Supported emulated systems and applications
- Supported emulated systems formats for the software
- Instructions on how to use the software

2.7 Assumptions and Dependencies

The team's assumption is that once our emulated MOS 6502 is fully functioning, the integration of code and the creation of another emulated system should be a straightforward process. However, as the implementation of the original processor varies from each individual hardware developer, this might be a more difficult process than it seems. This is because many of the chips have various clock speeds and memory mapping models. Our software will depend on the openGL rendering system to represent graphical output for the emulated systems that require them.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The software will prompt the user with a dialog in which he selects the machine he wants to emulate: Apple 1, Apple 2 or the NES. The user will be prompted to load additional software if required by the chosen machine. All the following GUI interactions will be dependent on the loaded software.

3.1.2 Hardware Interfaces

The software will interface with the hardware via standard peripherals, keyboard, mice and monitor. Additional functionality will be implemented via controller for use via NES or SNES games. The chosen game controller should have the same number of buttons or more as the chosen emulated system.

3.1.3 Software Interfaces

The initial version of the software will be command-line driven and will be runnable on most UNIX systems without any additional code bases or software requirements other than OpenGL 4.6. As development stages progress we expect to need additional code dependencies to seamlessly integrate our software with the chosen UNIX or Windows desktop environment, Mac, Debian, Windows, etc. Furthermore, we will need additional libraries for interfacing with an external peripheral such as a keyboard or controller.

3.1.4 Communications Interfaces

There are no external communications between our software and any other external machine. The MOS 6502 emulator is a self contained machine, the software running on the emulator requires no internet connection as it comes from a time before the Internet as we know it. There is no need for external encryption as the software does not store sensitive data.

We intend for our emulator to run on various CPUs and work flawlessly among them. As stated before, the software created for the MOS 6502 microprocessor depends on the internal clock speed of the microprocessor which varies based on the emulated system. Different CPUs with different execution times could lead to inconsistencies in with software execution. To rectify this discrepancy we must accurately time the execution of the emulated instruction as well as the execution time of the native MOS 6502 chip at a predetermined clock cycle. Put simply we count the time it takes for our CPU the execute the emulated instruction and then subtract that from the expected execution time of the MOS 6502 chip then we do nothing for the remainder of the time. For the host machine to run the emulator properly the length of execution time of any emulated instruction cannot exceed the execution time of the instruction on the native MOS 6502 chip.

3.2 Functional Requirements

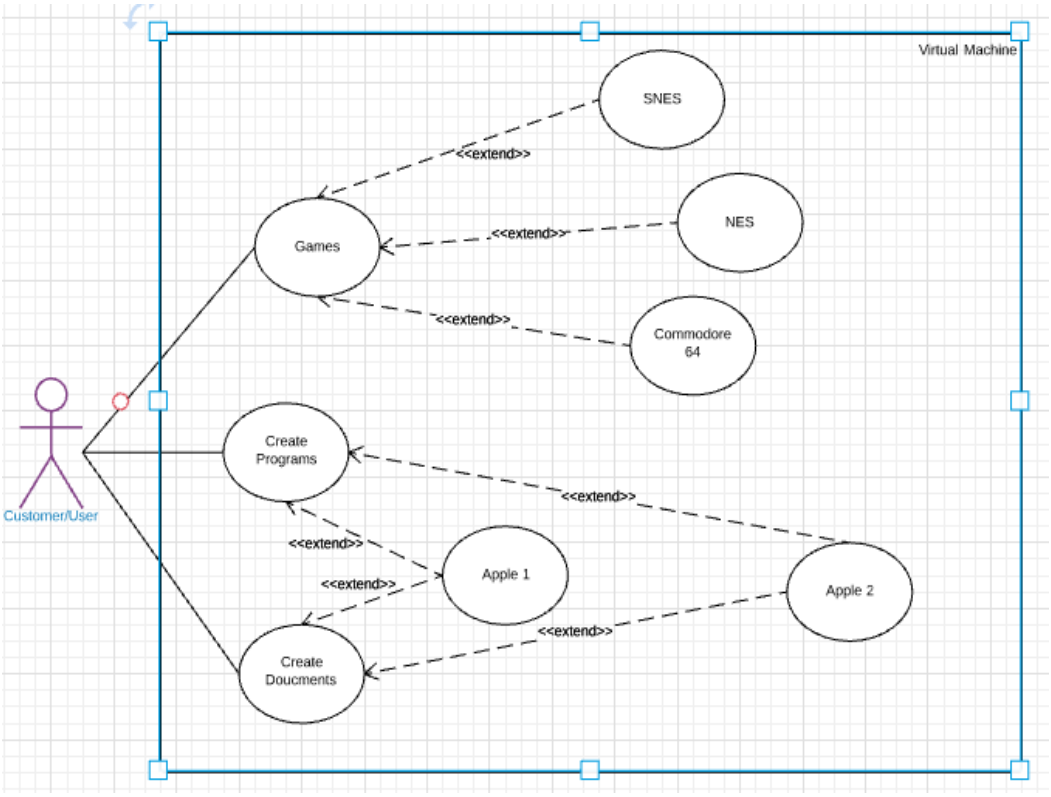
The initial goal of the emulator is to accurately emulate the instructions of the MOS 6502 microprocessor. Upon this rock we will build our church, meaning this is the most crucial part of our project and should behave flawlessly before we can attempt to accurately emulate a system of choice.

To accurately emulate a system, the emulated 6502 must be setup in accordance with the system model. The memory of an application must be accurately mapped on the emulated 6502 internal memory, as well as setting the registers to their proper initial states. Once a chosen MOS 6502 system is emulated, it is expected to run exactly as it would on its native microprocessor.

The emulator must also account for the device interfacing of every system. There must be a parser for the keyboard input to the emulated machine to achieve IO functionality. Furthermore, there should be an additional controller interfacing for systems that can use games controllers for a more intuitive experience.

3.3 Behaviour Requirements

3.3.1 Use Case View



4 Other Non-functional Requirements

4.1 Performance Requirements

The MOS 6502 emulated instructions should take exactly the same amount of time as they would a standalone 6502 chip for any given clock speed. All modern CPUs can execute these emulated instructions in a fraction of the time when compared with the 6502, even with the introduced overhead created by emulation. Once the instruction has been executed the application needs to remain idle in effort to preserve the timing constraints presented by the internal clock of the a MOS 6502, which is normally clocked at between 1MHZ and 3MHZ. Furthermore, there exist conditional clock cycles which must be accounted for. Branching instructions take an additional clock cycle to execute when a branch condition is taken and an additional cycle required if the branch crosses a page boundary. Memory indexing modes have similar pitfalls, when indexing memory that crosses a page boundary often requires an additional clock cycle to compute. Failing to account for these conditions will cause the application execution to be inconsistent in its timing, potentially causing random speed ups in time sensitive software.

4.2 Safety and Security Requirements

There are no additional security concerns with this software as it does not store sensitive data within the application or on the machine. Information that has been stored in an emulated machine is natively protected within the host operating system. If the machine running the emulator is password protected or encrypted access to the MOS 6502 emulator is extended by those safeguards. As the application does not connect to the internet there is no additional security concerns presented by the software.

4.3 Software Quality Attributes

The software is written in C and designed to be light and allow the software to be recompiled on multiple systems and CPU types. The software will be developed and tested modularly and integrated together as needed. Different systems have subsystems that must be implemented to achieve full system functionality. For instance the NES requires a secondary chip called the PPU (Picture Processing Unit) to render the visual output onto the screen. This modular systems must be implemented with full accuracy and integrated together seamlessly to emulate the system.

5 Other Requirements

There are no other requirements needed for this virtual machine. All requirements are covered in this Software Requirements Specifications document.

Appendix A – Data Dictionary

<Data dictionary is used to track all the different variables, states and functional requirements that you described in your document. Make sure to include the complete list of all constants, state variables (and their possible states), inputs and outputs in a table. In the table, include the description of these items as well as all related operations and requirements.>

There is currently no data dictionary used to track all the different variables, states, and functional requirements that was described in this document.

Appendix B - Group Log

<Please include here all the minutes from your group meetings, your group activities, and any other relevant information that will assist the Teaching Assistant to determine the effort put forth to produce this document>