1 Introduction

1.1 Intended Audience

The following document covers all functionality of the 6502 debugger and how each piece relates. This document is intended for programmers, team managers, and quality assurance on the developing team to ensure that the code is running to this specification.

1.2 How to Use this Document

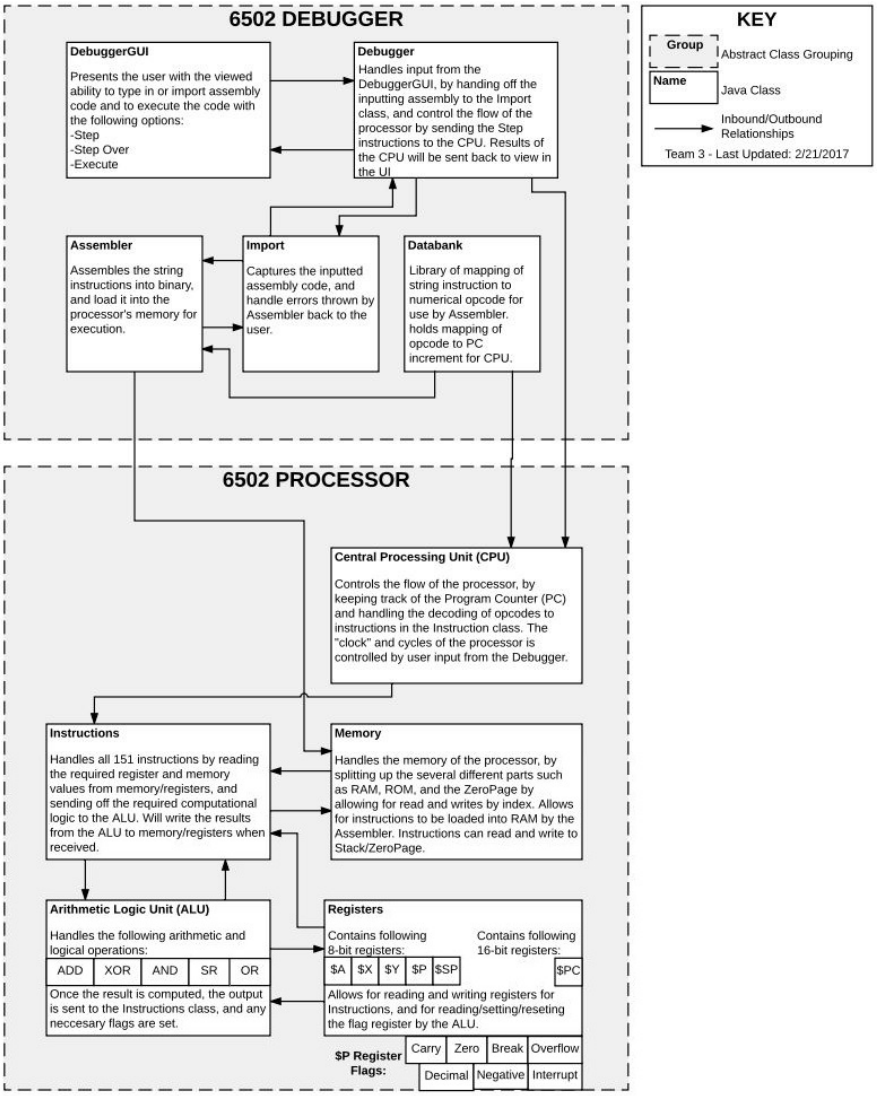
This document is intended to help organize all modules, classes, and functions found in the 6502 debugger. It is meant to help all team members design their components to this specification, as well as having a straightforward method of analyzing interactions between different components. Reasoning for why design choices have been made are found at the end of the document.

2 Summary

The program will be written in Java using JDK (Java Development Kit) 8. The program should accept any plain text input from the clipboard, manually typed in, or any file imported (regardless of validity). Due to the limited time frame of this project this is a living document and should be frequently referenced as specifications may change depending on customer requirements. This document will help create Quality Assurance documents needed to test the quality of the resulting application. Additionally, it will contain all the programming interactions between modules, classes, and functions. This will be achieved by both diagrams and showing all function headers and descriptions of what they achieve and their interactions.

Comment [AJL1]: Does the kind of file need to be specified? Should it say "any text file imported"

2.1 System Architecture


6502 DEBUGGER
DebuggerGUI
Presents the user with the viewed ability to type in or import assembly code and to execute the code with the following options:
-Step
-Step Over
-Execute
Debugger
Handles input from the DebuggerGUI, by handing off the inputting assembly to the Import class, and control the flow of the processor by sending the Step instructions to the CPU. Results of the CPU will be sent back to view in the UI
Assembler
Assembles the string instructions into binary, and load it into the processor's memory for execution.
Import
Captures the inputted assembly code, and handle errors thrown by Assembler back to the user.
Databank
Library of mapping of string instruction to numerical opcode for use by Assembler. holds mapping of opcode to PC increment for CPU.
6502 PROCESSOR
Central Processing Unit (CPU)
Controls the flow of the processor, by keeping track of the Program Counter (PC) and handling the decoding of opcodes to instructions in the Instruction class. The "clock" and cycles of the processor is controlled by user input from the Debugger.
Instructions
Handles all 151 instructions by reading the required register and memory values from memory/registers, and sending off the required computational logic to the ALU. Will write the results from the ALU to memory/registers when received.
Memory
Handles the memory of the processor, by splitting up the several different parts such as RAM, ROM, and the ZeroPage by allowing for read and writes by index. Allows for instructions to be loaded into RAM by the Assembler. Instructions can read and write to Stack/ZeroPage.
Arithmetic Logic Unit (ALU)
Handles the following arithmetic and logical operations:
ADD
XOR
AND
SR
OR
Once the result is computed, the output is sent to the Instructions class, and any neccesary flags are set.
Registers
Contains following 8-bit registers:
$A
$X
$Y
$P
$SP
Contains following 16-bit registers:
$PC
Allows for reading and writing registers for Instructions, and for reading/setting/reseting the flag register by the ALU.
$P Register Flags:
Carry
Zero
Break
Overflow
Decimal
Negative
Interrupt
KEY
Group
Abstract Class Grouping
Name
Java Class
Inbound/Outbound Relationships
Team 3 - Last Updated: 2/21/2017

3 User Interface Architecture

3.1 GUI Handbook

3.2 Expected Input

The user should be presented with the option to either load from file, paste from clipboard, or manually type in 6502 assembly instructions in an input-like window. The following are a list of valid 6502 codes accepted:

http://www.e-tradition.net/bytes/6502/6502_instruction_set.html

Other inputs are included as buttons which are accessed by mouse click in the main GUI window.

3.2.1 Output

The user should be presented both the memory output of the executed code in hex format as well as the current register values each in their own output window integrated as part of the main GUI screen. The customer has also indicated that they would like the line of code that was just executed to be highlighted in the input window with an indicator (arrow, bullet, asterisk, or other similar identifier) to mark the current line of code in the input window.

3.3 GUI Window

There will be an input window where the user inputs their code. Copy and paste functionality must be present in this window. Syntax highlighting for erroneous instructions must occur when the user tries to assemble the code and the results will appear in this window.

There will be a viewing window where contents of current memory will update at each stage of execution.

There will be a viewing window where the contents of the registers will update at each stage of execution.

There will be a file menu that will support opening a new session, loading a file to be parsed for 6502 assembly code, saving to that opened file or saving a new file of the current code, and finally an exit option.

3.3.1 Buttons

1. Step
2. Step Over
3. Execute
4. Memory Dump

3.4 Back-End Architecture

The 6502 processor is well defined and must be implemented/emulated according to the 6502 manual <<link>>.

4 Module/Class Specifications

This section details reasoning behind the design decisions of the 6502 debugger architecture. For a detailed specification of all modules and their function parameters please see the Function Interface Specifications (6502) sub section of the Software Design. The Function Interface Specification is intended for developers familiar with the Java programming language.

The system architecture is split into two main interfaces; the debugger and the processor. Since the 6502 requirements are well defined, the design decision was made to have the emulation of that processor have its own interface and once implemented should require little to no maintenance. The debugger interface however is where we have leeway to design how we see fit. We've broken down the interface trying to utilize a Model-View-Controller (MVC) design pattern in order to accomplish high cohesion and low coupling amongst models.

Comment [AJL2]: Rework this sentence.

The debugger interface links to the processor interface via three modules; Databank, Assembler, and the Debugger (Main control module). Databank services the translation of opcodes between the debugger and the CPU, Assembler feeds the Processor's Memory module, and the Debugger controls the message passing between the CPU and the User Interface.
<<Insert MVC diagram here>>

Comment [AJL3]: Since the debugger contains so many Java classes, does it need to be explained more in depth?

Even though the processor requirements are well defined there are some design choices for how we bring those requirements together. In the following text we will discuss how we designed the processor interface. The design will cover topics such as naming conventions, module linking, encapsulation, functionality, and of course reasoning why where sensible.

Deleted: togethers

Deleted: reasonings

<<Insert processor interface table here>>

5 Appendices