**Meeting Minutes**

**Date:** March 7, 2014

**Start Time:** 6:00pm

**End Time:** 9:00pm

**Members Present:** Drew Aaron, Michael Beaver, Clay Boren,

Chad Farley, Andrew Hamilton, and Travis Hunt

**Members Absent:** N/A

**Topics** **Discussed**

* Coding Standards
* Detailed Design

**Decisions and Actions Taken**

Andrew will draft the coding standards. He will present the coding standards to the team for modification and approval by March 11, 2014 but no later than March 13, 2014.

The team officially began formal discussions pertaining to the Detailed Design. The team has elected to begin with the Backend since it requires the most effort to design and implement. The Backend has been logically divided into six interacting modules: Error Detection; Symbol Table; Translator; Library; Drivers; and, Memory. The Drivers module is further subdivided into the Assembler and the Simulator. The Error Detection module is further subdivided into sections for the Assembler and the Simulator. The Memory module is further subdivided into sections for Registers and Main Memory, both for the Simulator. The team has not yet begun the Detailed Design process for the Assembler, the Simulator, and the Memory modules. Note that all module names herein are subject to change. See the attached graphics.

The Error Detection module consists of an abstract base class ErrorDetection and two child classes AssemblerErrors and SimulatorErrors. The AssemblerErrors and the SimulatorErrors classes have similar functionality but behave differently. Hence, the ErrorDetection base class has virtual functions that may be overwritten in the child classes. Two of these functions include the errorHandler() and the errorReport() functions. Each child class also needs its own specific errorLibrary[] data member.

The Symbol Table module is constituted by the SymbolTable class. The SymbolTable class will be made up of the symbolTable data member, most likely a 2-D array representation of a hash table. The symbolTable will contain symbols as well as literals to save space. To maintain a certain level of efficiency, the symbolTable data member will have a limited size. The SQA group has determined that a standard CS 310 program ought not to exceed 100 symbols and 25 literals. Hence, the symbolTable will have 251 hash slots.[[1]](#footnote-1) Note, however, that the numerical limits on symbols and literals will be set via constants that may be updated as necessary. The exact hash function and collision handling will be determined at a later date. The SymbolTable class will also have all the relevant accessor and mutator methods, such as getSymbol() and hashSymbol().

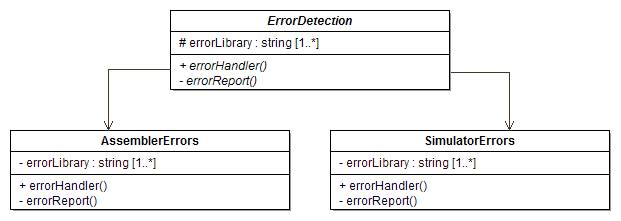
The Translator is constituted by the abstract base class Translator. The child classes Pass1Translator and Pass2Translator inherit from the Translator base class and correspond to the respective passes of the Assembler. The Translator class and its children essentially act as a collection of functions to be used by the Assembler. The Translator base class will contain any common virtual methods that may be overwritten by the child classes. The Pass1Translator will handle—but is not limited to—any symbol table, location counter, and boundary alignment calculations and updates. The Pass2Translator will handle—but is not limited to—any .PRT generation, object code generation, and error reporting to the Error Detection module.

The Library module is constituted by the Library class, which has a Definitions class. The Library class will maintain a hash table of the machine operations. It will also have the functionality to lookup into the machine operations table and determine whether a given operation is indeed a valid operation. The Definitions class will have the functionality for each instruction.

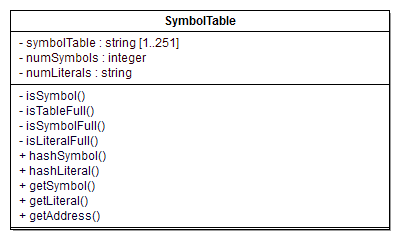
The team will work on refining the Detailed Design at the next meeting. The next team meeting will be at 10:00am on March 8, 2014 at the Christian Student Center.

**Supplementary Information**

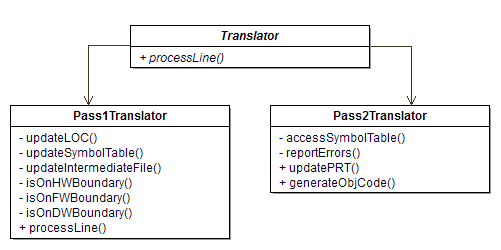
**Error Detection Module**

****

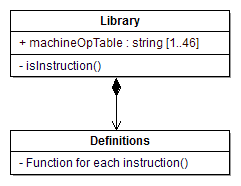
**Symbol Table Module**

****

**Translator Module**

****

**Library Module**

****

1. (100 + 25) x 2 = 250 + 1 = 251→ prime number. [↑](#footnote-ref-1)