Concordia University Department of Computer Science and Software Engineering

SOEN 341 Winter 2021 - Project - Final Report

Presented to Dr. Michel de Champlain

Team 9 - Far From Home

Yushan Yang - 40099151
Maya McRae - 27536143
Mohona Mazumdar - 40129421
Jasmine Lebel - 40135464
Marita Brichan - 40138194
Gechen Ma - 40026175
Tarun Elango - 40084007
Aida Kordi - 40045621
Yu Fei Xiang - 40089232

Table of Contents

List of Contents

List of Figures

List of Tables

Introduction

The purpose of this project is to construct, using the Java programming language, a cross-assembler that processes Cm Assembly Language, all the while applying agile processes learned in the course. The cross-assembler reads an assembly language source file to generate an in-memory intermediate representation (IR), which is then traversed, outputting an executable file. The assembly language is a source file that consists of low-level programming, a series of binary instructions that has one to one correspondence with machine code, and is represented by the following symbol names: labels, mnemonics and operands. The assembly language source files are provided to the team.

A cross-assembler is a software system composed of help, verbose, and listing options. It generates a source listing and a label table after pass 1, and it generates errors that do not follow EBNF grammar. In total, an assembler will make two passes. In the first pass, the assembly language code will be traversed by the assembler in order to generate the instructions, such as the mnemonics, labels, operands, etc., that will comprise the machine learning code. Furthermore, the assembler will generate a symbol table, where a label will be related to an offset within a particular instruction, will generate any offset that can be resolved, and will complete the first pass by indicating any offsets that will need to be resolved in pass 2. In the second pass, the assembler will then traverse the sequence of instructions in order to set any non-resolved offsets.

Sprint 1

For the project's first sprint, the team was tasked with creating a domain dictionary of the 15 most important concepts and operations of a cross-assembler. The goal of this sprint was to get familiar with the initial phase of an agile software process, where we were able to observe and describe, from a high-level point of view, the software system's requirements in order to adequately develop a cross-assembler. The domain dictionary can be viewed below:

Domain Dictionary

Concept: **assembly language source file**
Definition: Source file that is a symbolic, one-to-one representation of a corresponding machine language, which the lexer must read form.
Modeled as: ----
System Name: .asm

Concept: **code generator**
Definition: Traverses an intermediate representation and generates an executable file and a listing file.
Modeled as: Method
System Name: generateLstFile, generateExeFile

Concept: **command-line**
Definition: An interface that gives the user options to assemble a Cm assembly program .asm file. The interface is navigated by the user typing commands at prompts.
Modeled as: Class
System Name: CommandLine

Concept: **cross-assembler**
Definition: Translates assembly language source file into a machine language executable file.
Modeled as: Package
System Name: CrossAssembler

Concept: **error reporter**
Definition: Reports errors by saving the error and token position and outputting each line that has an error.
Modeled as: Class
System Name: ErrorReporter

Concept: **executable file**
Definition: Output of the cross-assembler.
Modeled as: .exe file
System Name: output.exe

Concept: **instruction**
Definition: Consists of an operation code immediately followed by operands (or instruction parameters).

Modeled as: Class
System Name: Instruction

Concept: **intermediate representation**
Definition: Composed of three optional parts (label, instruction/directive, and a comment) followed by an end-of-line (EOL).
Modeled as: Class
System Name: IR

Concept: **lexer**
Definition: Imports data from an assembly language source file until labels, instructions or comments identified. It reports errors.
Modeled as: Class
System Name: Lexer

Concept: **listing file**
Definition: Text file containing a list of data. Output of the cross-assembler.
Modeled as: .lst file
System Name: output.lst

Concept: **parser**
Definition: Parses tokens and generates an intermediate representation. It reports errors.
Modeled as: Class
System Name: Parser

Concept: **reader**
Definition: Reads the assembly source file.
Modeled as: Class
System Name: Reader

Concept: **symbol table**

Definition: Maps each mnemonic with a key and value in a table format. Key is the mnemonic string, value is the mnemonic object representing the String name and its opcode.

Modeled as: Hash table
System Name: symbolTable

Concept: **token**
Definition: Can be mnemonics, labels, offsets, or comments.
Modeled as: Class
System Name: Token

Concept: **user**
Definition: Enters commands on command-line, and provides the source file.
Modeled as: ----
System Name: ----

Following the domain dictionary, the second expected deliverable for sprint 1 was a key concept model of the cross-assembler. The key concept model can be viewed below:

<u>Key Concept Model of the Cm Cross-Assembler</u>

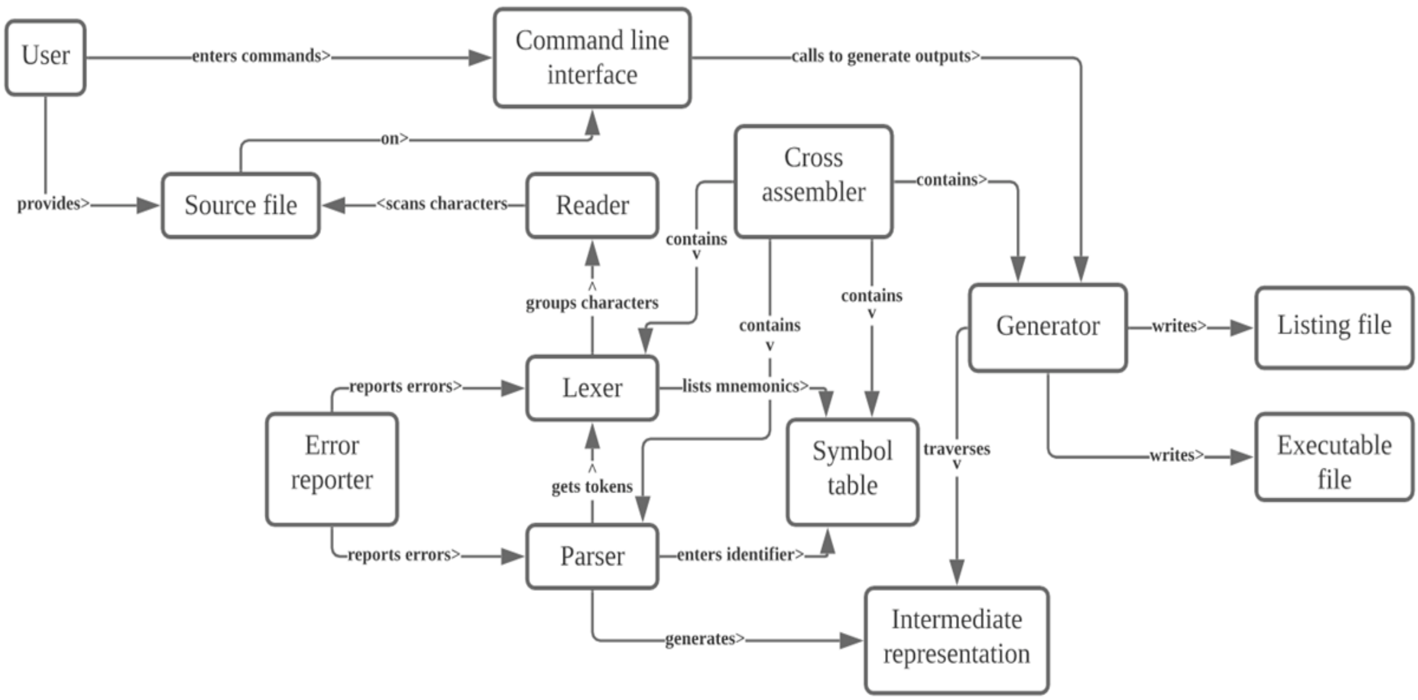

Figure 1. Key Concept Model of the Cm Cross-Assembler

Lastly, the third expected deliverable for sprint 1 was a use case UML diagram, extracting the cross-assembler's seven essential functionalities, excluding the scenarios. The use case diagram can be viewed on the following page:

Use Case Diagram of the Cm Cross-Assembler

Figure 2. Use Case Diagram of the Cm Cross-Assembler

Sprint 2

The main goal of sprint 2 was to design, implement, and to test all the stories extracted in the product backlog. In short, the team was able to generate a listing file for inherent instructions, read a given assembly source file, and scan characters to create and parse tokens to create an intermediate representation. The simplified class diagram for the second sprint can be viewed below:

<u>Simplified Class Diagram for Sprint 2</u>

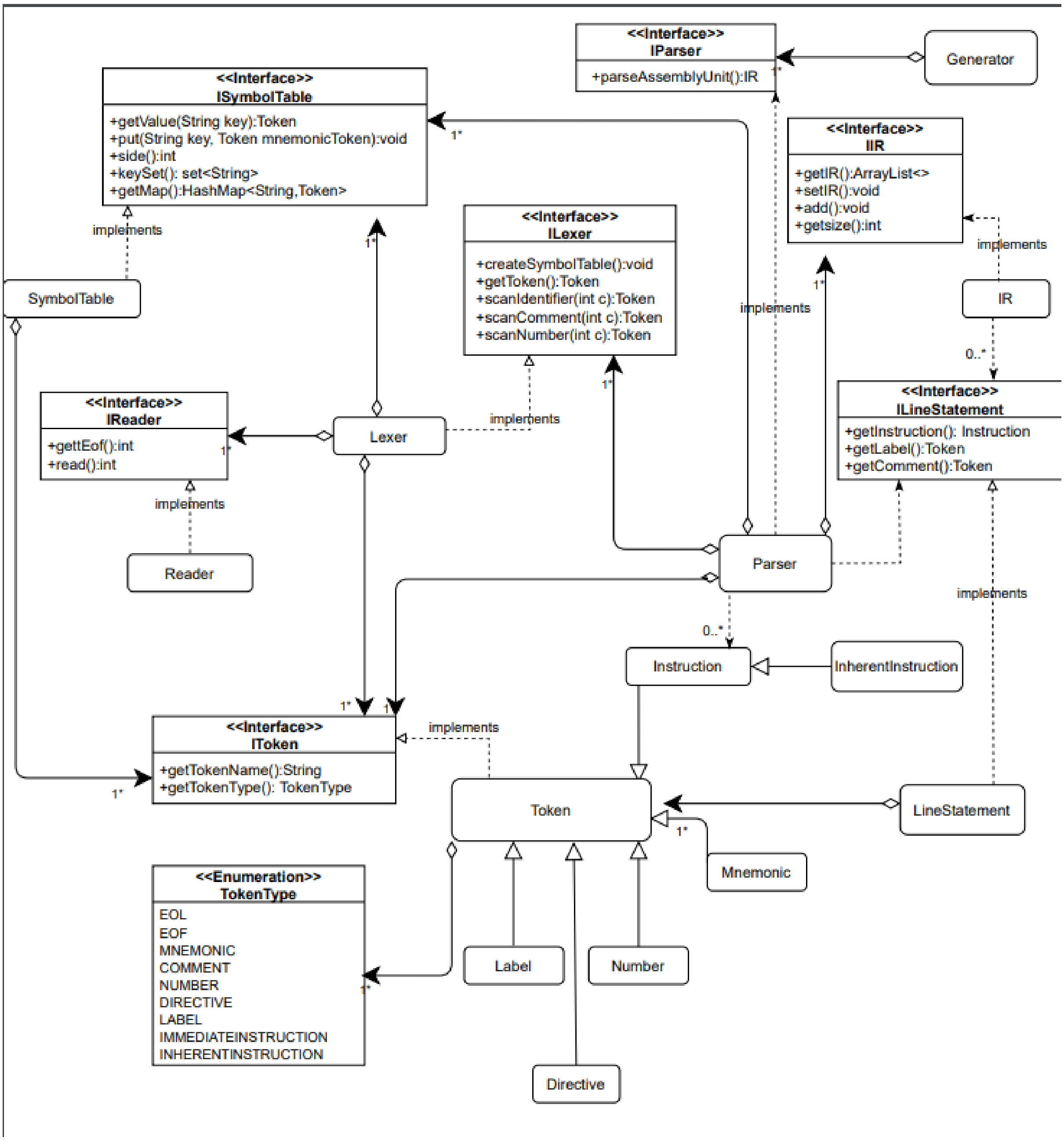

Figure 3. Simplified Class Diagram for Sprint 2

Estimation Velocity versus Actual Velocity for high-priority (red) items

Sprint	Finished/In Progress/Not Started	Estimated Velocity	Actual Velocity
[2]	Finished	2	2
[3]	Finished	3	3
[4]	Finished	4	4
[7]	Finished	7	7
[1]	Not Started		
[6]	Not Started		
[5]	Not Started		

Table 1. Estimated Velocity vs. Actual Velocity Table for Sprint 2

Sprint 3

For Sprint 3, the team was focused on testing all the stories from the second sprint product backlog as well as generating a listing file (this time for immediate instructions), scanning comments and numbers to create the tokens, parsing the rest of the tokens (instructions, directives and comments) to generate the immediate instruction and reporting errors by creating the error reporter. The simplified class diagram for the third sprint can be viewed below:

<u>Simplified Class Diagram for Sprint 3</u>

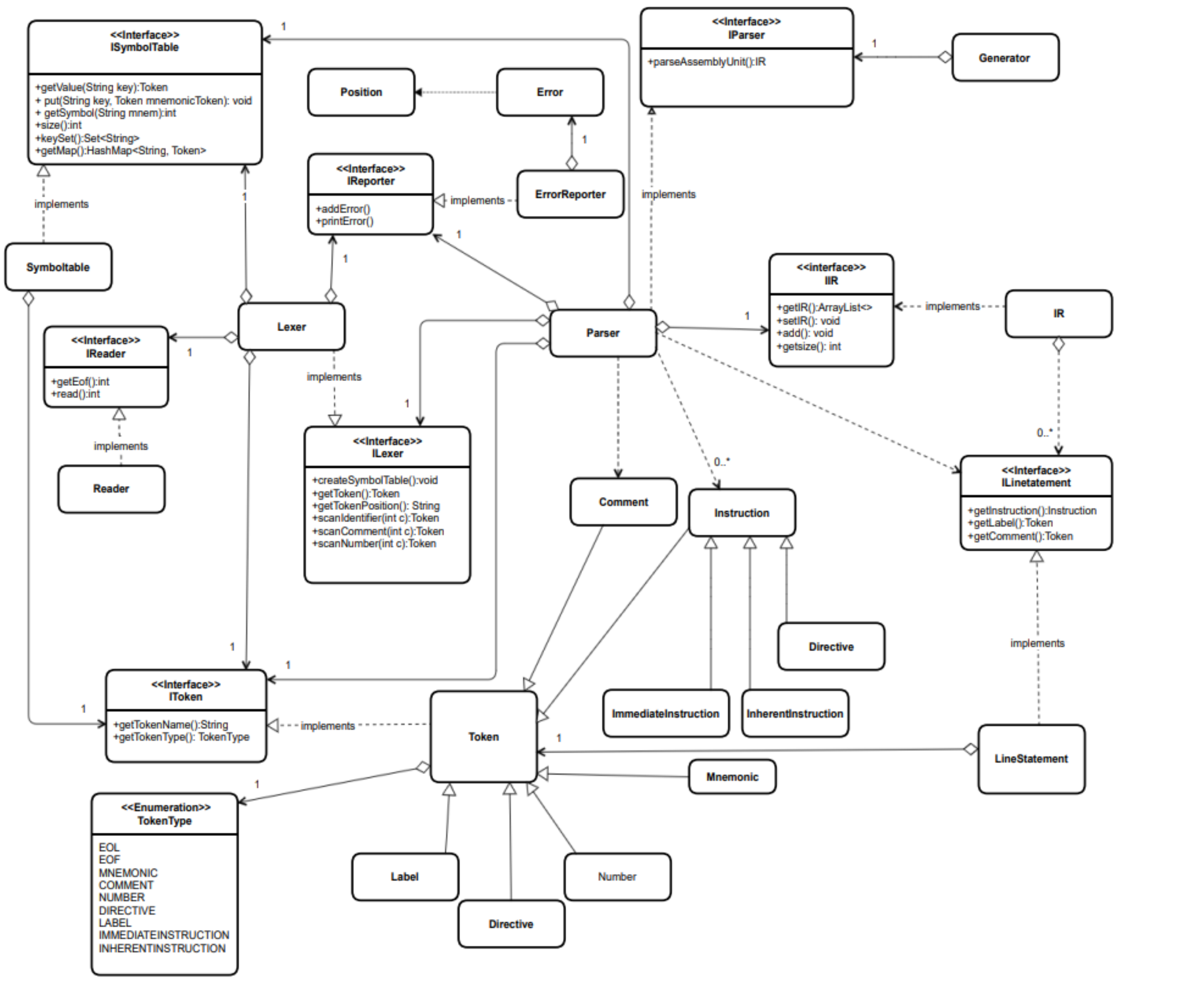

Figure 4. Simplified Class Diagram for Third Sprint

Estimation Velocity versus Actual Velocity for high-priority (red) items

Story	Status	Estimated Velocity	Actual Velocity
[3]	Finished	5	8
[4]	Finished	8	13
[5]	Finished	8	8
[7]	Finished	13	5
[1]	Not Started		
[6]	Not Started		

Table 2. Estimated Velocity vs. Actual Velocity Table for Sprint 3

Sprint 4

For the fourth and final sprint, the goal was to fully complete the project of the cross-assembler. The team was tasked with generating a listing file for inherent, immediate and relative instructions, generating an executable file, parsing options and the source file on the command line, and integrating an abstract factory as well as a builder design patterns for creating the IR.

<u>Simplified Class Diagram for Sprint 4</u>

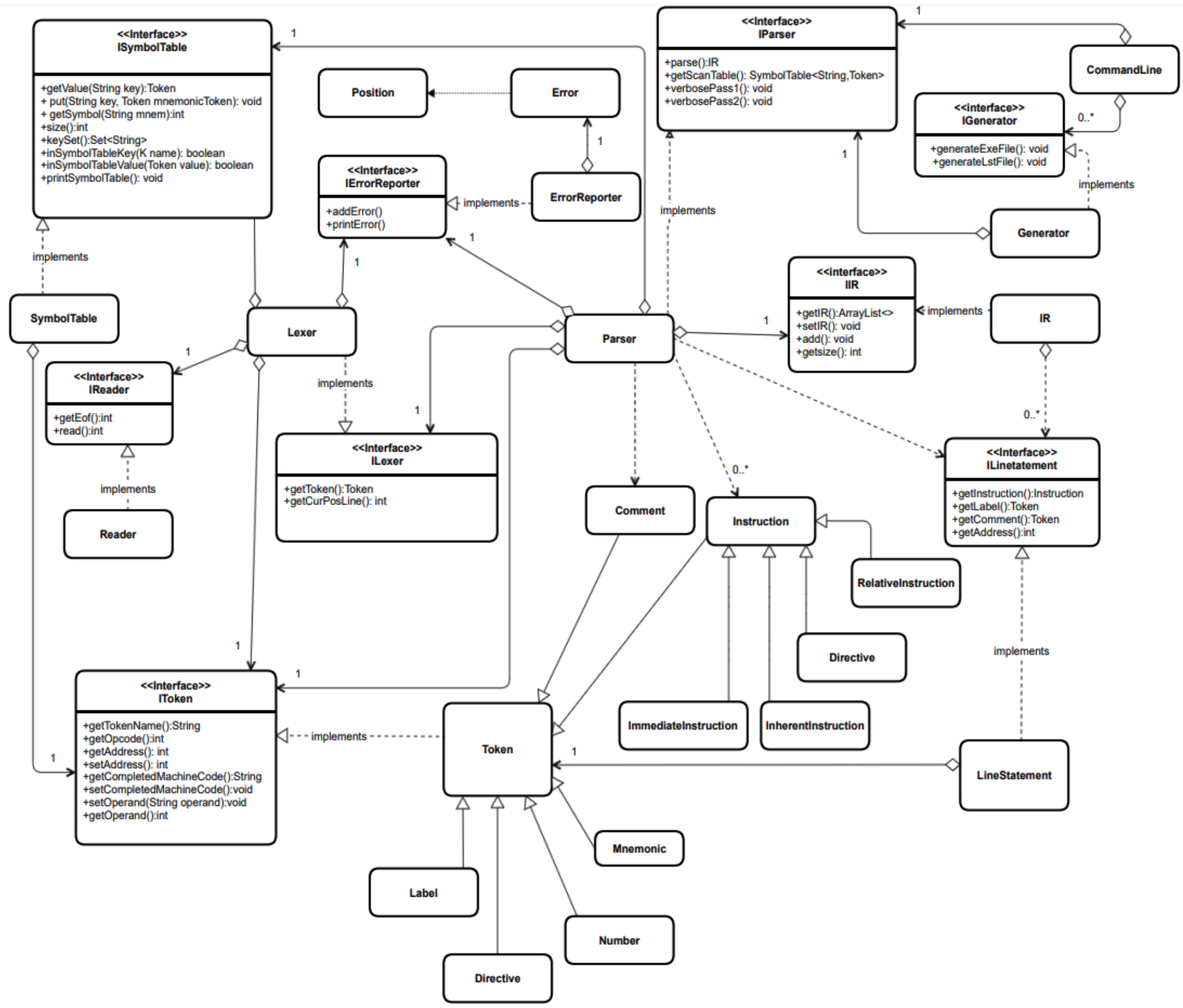

Figure 5. Simplified Class Diagram for Fourth Sprint

Estimation Velocity versus Actual Velocity for high-priority (red) items

Story	Status	Estimated Velocity	Actual Velocity
[1]	Finished	8	8
[4]	Finished	5	13
[6]	Finished	13	5
[7]	Finished	1	1

Table 3. Estimated Velocity vs. Actual Velocity Table for Sprint 4

Road Map

Product Backlog for Sprint 2:

Sprint Backlog 2									
Epic 1 - Parse options and the source file on the command-line									
Stories	Story Name	Status	Priority	Due Date	Start Date	End Date	Time Estimate	Confidence Level	Progress
story a	Enter valid options in a cache (at init)	Backlog	LOW						
story b	Validate and enable options	Backlog	LOW						
story c	Validate the source file and check if the file exists	Backlog	LOW						
Epic 2 - Read the assembly source file									
Stories	Story Name	Status	Priority	Due Date	Start Date	End Date	Time Estimate	Confidence Level	Progress
story a	open source file in read mode	Completed	HIGH	25-02-2021	16-02-2021	17-02-2021	1h	On track	100%
story b	create and open a file input stream	Completed	HIGH	25-02-2021	16-02-2021	17-02-2021	1h	On track	100%
Epic 3 - Scan characters to extract and create tokens									
Stories	Story Name	Status	Priority	Due Date	Start Date	End Date	Time Estimate	Confidence Level	Progress
story a	Create a symbol table (for labels and mnemonics)	Completed	HIGH	25-02-2021	17-02-2021	18-02-2021	2h	On track	100%
story b	Enter mnemonics (keywords) in the symbol table	Completed	HIGH	25-02-2021	17-02-2021	18-02-2021	2h	On track	100%
story c	Read characters from the input stream	Completed	HIGH	25-02-2021	17-02-2021	18-02-2021	2h	On track	100%
story d	Keep track of the position (line, column) of the token	Backlog	LOW						
story e	Skip white spaces	Completed	HIGH	25-02-2021	17-02-2021	18-02-2021	1h	On track	100%
story f	Scan comments	Backlog	LOW						
story g	Scan identiers (no labels yet, only mnemonics)	Completed	HIGH	25-02-2021	18-02-2021	18-02-2021	3h	On track	100%
story h	Scan numbers	Backlog	LOW						
story i	Report scanner errors	Backlog	LOW						
story j	Return token to parser	Completed	HIGH	25-02-2021	18-02-2021	18-02-2021	2h	On track	100%
Epic 4 - Parse tokens and generate an IR									
Stories	Story Name	Status	Priority	Due Date	Start Date	End Date	Time Estimate	Confidence Level	Progress
story a	Create an IR (array or list of line statements)	Completed	HIGH	25-02-2021	20-02-2021	20-02-2021	2h	On track	100%
story b	Parse an assembly unit	Completed	HIGH	25-02-2021	20-02-2021	20-02-2021	2h	On track	100%
story c	Parse a line statement and add it to IR	Completed	HIGH	25-02-2021	20-02-2021	21-02-2021	3h	On track	100%
story d	Parse a label	Backlog	LOW						
story e	Parse a instruction (mnemonic and operand)	Backlog	LOW						
story f	Parse a directive	Backlog	LOW						
story g	Parse a comment	Backlog	LOW						
story h	Parse an end of line	Completed	HIGH	25-02-2021	20-02-2021	21-02-2021	2h	On track	100%
story i	Parse an end of file and return the IR to the back end	Completed	HIGH	25-02-2021	20-02-2021	21-02-2021	3h	On track	100%
Epic 5 - Report errors to the error reporter									
story a	Save the error (number or message) and token's position	Backlog	MEDIUM						
story b	Record (or enter) the error in an ordered list of errors	Backlog	MEDIUM						
story c	Print each line that has an error	Backlog	MEDIUM						
Epic 6 - Generate an executable file									
story a	Traverse the IR using the symbol table	Backlog	LOW						
story b	Find the instructions that are not resolved	Backlog	LOW						
story c	Create an executable file	Backlog	LOW						
story d	Write all the binary code from a code buffer to the executable file	Backlog	LOW						
Epic 7 - Generate a listing file									
Stories	Story Name	Status	Priority	Due Date	Start Date	End Date	Time Estimate	Confidence Level	Progress
story a	Traverse the IR using the symbol table	Completed	HIGH	25-02-2021	20-02-2021	21-02-2021	2h	On track	100%
story b	Create the format header of a line statement	Completed	HIGH	25-02-2021	22-02-2021	23-02-2021	3h	On track	100%
story c	Create the inherent instruction formatter	IN PROGRESS	HIGH	25-02-2021	22-02-2021	24-02-2021	3h	Might be late	60%
story d	Create the immediate instruction formatter	Backlog	LOW						
story e	Create the relative instruction formatter	Backlog	LOW						
story f	Generate the opening of a line statement in a listing file	IN PROGRESS	HIGH	25-02-2021	23-02-2021	24-02-2021	2h	Might be late	70%
story g	Generate the closing of a line statement in a listing file	Completed	HIGH	25-02-2021	23-02-2021	24-02-2021	2h	On track	100%

Table 4. Product Backlog for Sprint 2

Product Backlog for Sprint 3:

Sprint Backlog 3									
Epic 1 - Parse options and the source file on the command-line									
Stories	Story Name	Status	Priority	Due Date	Start Date	End Date	Time Estimate	Confidence Level	Progress
story a	Enter valid options in a cache (at init)	Backlog	LOW						
story b	Validate and enable options	Backlog	LOW						
story c	Validate the source file and check if the file exists	Backlog	LOW						
Epic 3 - Scan characters to extract and create tokens									
Stories	Story Name	Status	Priority	Due Date	Start Date	End Date	Time Estimate	Confidence Level	Progress
story d	Keep track of the position (line, column) of the token	Completed	HIGH	25-03-2021	19-03-2021	20-03-2021	3h	On track	100%
story f	Scan comments	Completed	HIGH	25-03-2021	19-03-2021	22-03-2021	4h	On track	100%
story h	Scan numbers	Completed	HIGH	25-03-2021	20-03-2021	23-03-2021	2h	On track	100%
story i	Report scanner errors	Completed	HIGH	25-03-2021	20-03-2021	23-03-2021	2h	On track	100%
Epic 4 - Parse tokens and generate an IR									
Stories	Story Name	Status	Priority	Due Date	Start Date	End Date	Time Estimate	Confidence Level	Progress
story d	Parse a label	Backlog	LOW						
story e	Parse a instruction (mnemonic and operand)	Completed	HIGH	25-03-2021	19-03-2021	25-03-2021	5h	On track	100%
story f	Parse a directive	Completed	HIGH	25-03-2021	19-03-2021	21-03-2021	4h	On track	100%
story g	Parse a comment	Completed	HIGH	25-03-2021	21-03-2021	23-03-2021	3h	On track	100%
Epic 5 - Report errors to the error reporter									
story a	Save the error (number or message) and token's position	Completed	HIGH	25-03-2021	19-03-2021	20-03-2021	1h	On track	100%
story b	Record (or enter) the error in an ordered list of errors	Completed	HIGH	25-03-2021	19-03-2021	22-03-2021	3h	On track	100%
story c	Print each line that has an error	Completed	HIGH	25-03-2021	20-03-2021	22-03-2021	2h	On track	100%
Epic 6 - Generate an executable file									
story a	Traverse the IR using the symbol table	Backlog	MEDIUM						
story b	Find the instructions that are not resolved	Backlog	MEDIUM						
story c	Create an executable file	Backlog	MEDIUM						
story d	Write all the binary code from a code buffer to the executable file	Backlog	MEDIUM						
Epic 7 - Generate a listing file									
Stories	Story Name	Status	Priority	Due Date	Start Date	End Date	Time Estimate	Confidence Level	Progress
story c	Create the inherent instruction formatter	IN PROGRESS	HIGH	25-02-2021	19-02-2021	22-02-2021	3h	On track	100%
story d	Create the immediate instruction formatter	Completed	HIGH	25-03-2021	22-02-2021	23-02-2021	4h	On track	100%
story f	Generate the opening of a line statement in a listing file	IN PROGRESS	HIGH	25-02-2021	19-02-2021	22-02-2021	3h	On track	100%

Table 5. Product Backlog for Sprint 3

Product Backlog for Sprint 4:

Sprint Backlog 4									
Epic 1 - Parse options and the source file on the command-line									
Stories	**Story Name**	**Status**	**Priority**	**Due Date**	**Start Date**	**End Date**	**Time Estimate**	**Confidence Level**	**Progress**
story a	Enter valid options in a cache (at init)	Completed	HIGH	23-04-2021	10-04-2021	20-04-2021	3h	On track	100%
story b	Validate and enable options	Completed	HIGH	23-04-2021	10-04-2021	21-04-2021	2h	On track	100%
story c	Validate the source file and check if the file exists	Completed	HIGH	23-04-2021	10-04-2021	21-04-2021	3h	On track	100%
Epic 4 - Parse tokens and generate an IR									
Stories	**Story Name**	**Status**	**Priority**	**Due Date**	**Start Date**	**End Date**	**Time Estimate**	**Confidence Level**	**Progress**
story d	Parse a label	Completed	HIGH	23-04-2021	12-04-2021	14-04-2021	2h	On track	100%
story e	Parse a instruction (mnemonic and operand)	Completed	HIGH	23-04-2021	19-04-2021	21-04-2021	2h	On track	100%
Epic 6 - Generate an executable file									
story a	Traverse the IR using the symbol table	Completed	HIGH	23-04-2021	10-04-2021	10-04-2021	1h	On track	100%
story b	Find the instructions that are not resolved	Completed	HIGH	23-04-2021	15-04-2021	20-04-2021	3h	On track	100%
story c	Create an executable file	Completed	HIGH	23-04-2021	10-04-2021	20-04-2021	1h	On track	100%
story d	Write all the binary code from a code buffer to the executable file	Completed	HIGH	23-04-2021	10-04-2021	20-04-2021	3h	On track	100%
Epic 7 - Generate a listing file									
Stories	**Story Name**	**Status**	**Priority**	**Due Date**	**Start Date**	**End Date**	**Time Estimate**	**Confidence Level**	**Progress**
story e	Create the relative instruction formatter	Completed	HIGH	23-04-2021	20-04-2021	22-04-2021	2h	On track	100%

Table 6. Product Backlog for Sprint 4

Sequence diagram

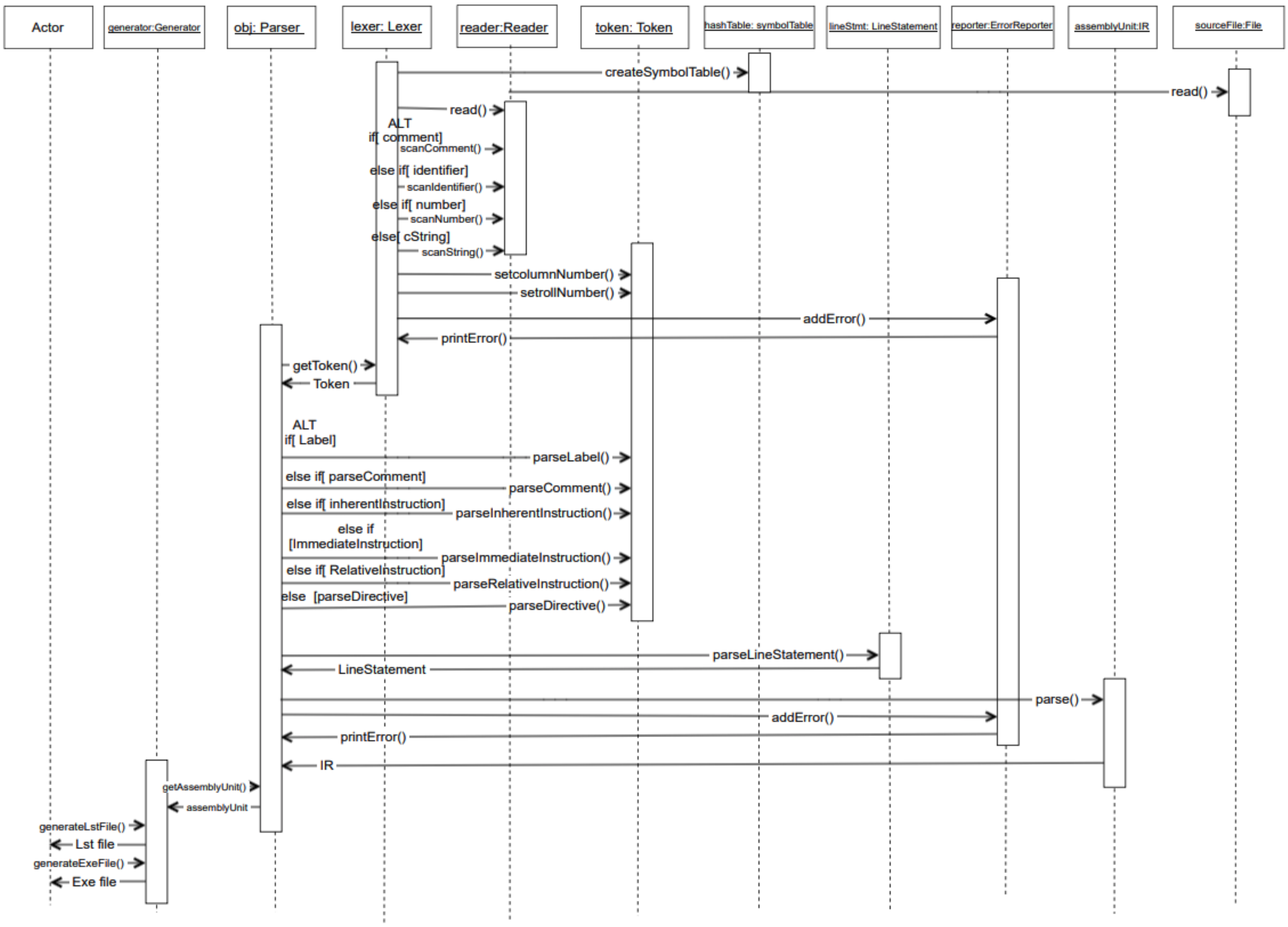

Figure 6. Sequence Diagram

Overall Architecture

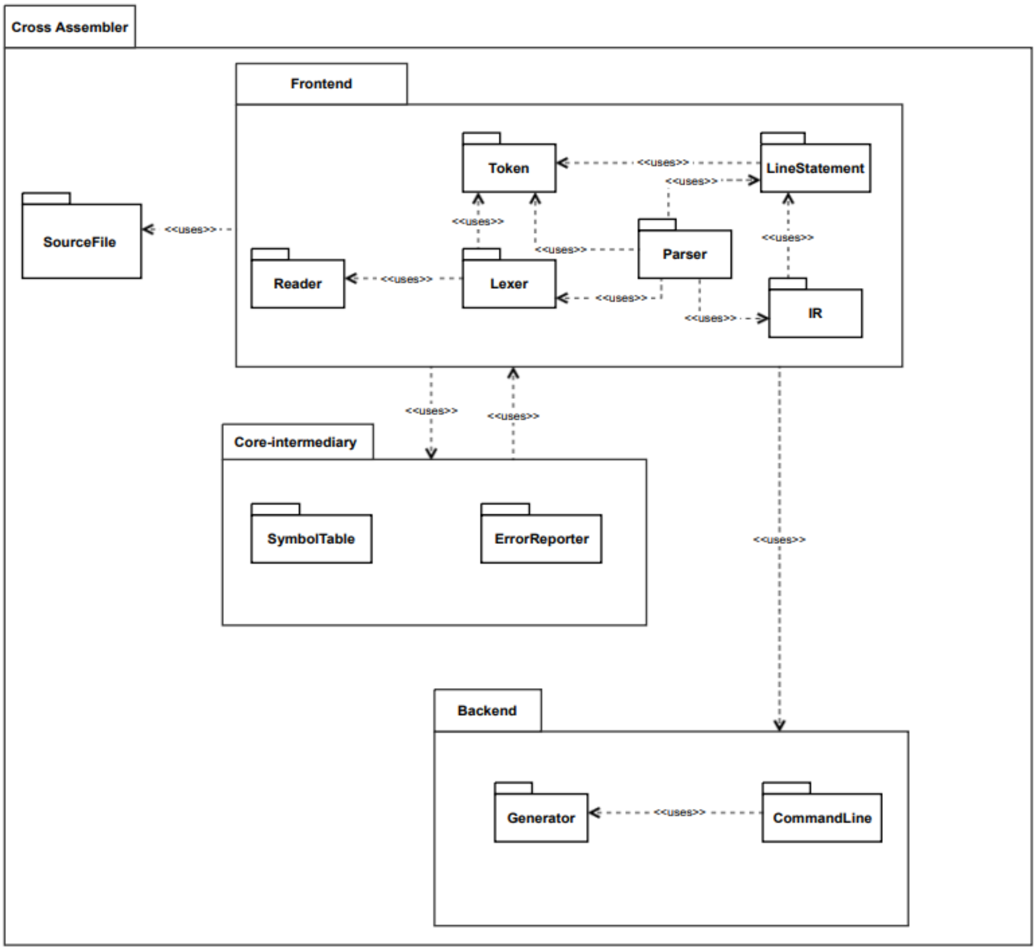

Figure 7. Overall Architecture of the Cross-Assembler

Discussion

Throughout the entire project, the team has attempted to understand the agile software development process, particularly the Scrum process, and apply them in a practical setting. This was done by going over the product backlog multiple times and prioritizing the right items for each sprint, assigning different tasks to each development team member, organizing standups (daily scrums) where the team stays connected and informed for the duration of the project and planning sprint reviews after each sprint to assess the team's strengths and weaknesses among all the stories of the sprint. The Scrum process has allowed the team to collaborate at all times, despite being a large software development team. One issue we have encountered during the project is setting realistic goals and accomplishing each story in the given timeframes. In fact, there were times where the estimation velocities for a sprint were far off from the actual velocities. The team would sometimes underestimate the time needed to complete a story which resulted in poor time management for the corresponding story. Lastly, we have learned that it is important to design code that can be tested throughout the development process by using unit testing and refactoring the code frequently during small tests.

Conclusion

To conclude, the purpose of this project was to, over the course of around 15 weeks, construct a cross-assembler that processes Cm Assembly Language. This project enabled us to, as a team, apply agile software processes in order to develop a software application. Throughout the four sprints, we were able to completely develop a cross-assembler capable of reading and writing files in assembly language. Furthermore, we were familiarized with using building blocks of object-oriented modelling, enabling us to create different types of diagrams, such as UML's of key concept models or use case diagrams, as well as class diagrams. We were able to enter the world of agile processing, and experience first-hand what it's like to deal with epics, user stories, sizing product backlog items with the planning poker technique, as well as testing and refactoring our code over the course of four different sprints.

List of Java files

- CommandLine.java
- Comment.java
- Directive.java
- endOfLine.java
- Error.java
- ErrorReporter.java
- Generator.java
- IErrorReporter.java
- IIR.java
- ILexer.java
- ILineStatement.java
- ImmediateInstruction.java
- InherentInstruction.java
- Instruction.java
- IParser.java
- IPosition.java
- IR.java
- IReader.java
- ISymbolTable.java
- IToken.java
- Label.java
- Lexer.java
- LineStatement.java
- Mnemonic.java
- Number.java
- Parser.java
- Position.java
- Reader.java
- RelativeInstruction.java
- SymbolTable.java
- TestErrorReporter.java
- TestGenerator.java
- TestIR.java
- TestLexer.java
- TestLineStatement.java
- TestParser.java
- TestReader.java
- TestSymbolTable.java
- TestToken.java
- Token.java