**ASSIST/UNA: A Windows-Based Emulator**

**of the ASSIST/I Assembler for the IBM/360**

Software Requirements Specification

Version 1.0.3

February 8, 2014

Travis Hunt, *Team Leader*

Michael Beaver, *Technical Writer*

Andrew Hamilton, *Software Quality Assurance*

Drew Aaron

Clay Boren

Chad Farley

University of North Alabama

CS 455: Software Engineering

Dr. Patricia L. Roden

Spring 2014

# Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version Number** | **Author(s)** | **Comments** |
| 02/05/2014 | Version 1.0 | Michael Beaver | Initial draft |
| 02/06/2014 | Version 1.0.1 | Michael Beaver | Added definitions, acronyms, and abbreviations;  Added project scope;  Added references;  Added user characteristics;  Added change management process; Added assembly instructions appendix;  Removed unnecessary sections |
| 02/07/2014 | Version 1.0.2 | Michael Beaver | Added product perspective;  Added product functions;  Added assumptions and dependencies;  Added functional requirements;  Added use cases;  Added design constraints;  Added other requirements;  Added diagrams to appendix;  Added prototype screen captures;  Updated definitions, acronyms, and abbreviations;  Updated references;  Removed unnecessary sections |
| 02/08/2014 | Version 1.0.3 | Michael Beaver | Added introductory overview;  Added general constraints;  Added external interface requirements;  Added classes/objects;  Added future features appendix;  Updated references;  Updated assembly instructions appendix |

**TO DO:** ASSIST is the original assembler and ASSIST/I is the DOS version.

**TO DO:** “source program code” to “program source code.”

**Table of Contents**

Version History i

1. Introduction 1

1.1 Purpose 1

1.2 Scope 1

1.3 Definitions, Acronyms, and Abbreviations 1

1.4 References 3

1.5 Overview 3

2. General Description 4

2.1 Product Perspective 4

2.2 Product Functions 4

2.3 User Characteristics 4

2.4 General Constraints 4

2.5 Assumptions and Dependencies 4

3. Specific Requirements 6

3.1 External Interface Requirements 6

3.1.1 User Interfaces 6

3.1.2 Software Interfaces 7

3.2 Functional Requirements 7

3.2.1 Source Code Editing 8

3.2.2 Source Code Assembly 8

3.2.3 Source Code Assembly and Debugging 9

3.2.4 Source Code Assembly and Final Run 9

3.2.5 Report Viewing and Printing 10

3.3 Use Cases 10

3.3.1 Create a New Project 10

3.3.2 Save a Project 10

3.3.3 Open a Project 11

3.3.4 Assemble a Program 11

3.3.5 Assemble a Program and Debug 11

3.3.6 Assemble a Program and Perform a Final Run 12

3.3.7 View a .PRT File 12

3.3.8 Print a .PRT File 12

3.3.9 Print Source Program Code 13

3.4 Classes / Objects 13

3.4.1 Backend Classes 13

3.4.2 Frontend Classes 14

3.5 Non-Functional Requirements 14

3.5.1 Performance 15

3.5.2 Reliability 15

3.5.3 Availability 15

3.5.4 Security 15

3.5.5 Maintainability 15

3.5.6 Portability 15

3.6 Design Constraints 15

3.7 Other Requirements 15

4. Deliverables 16

5. Change Management Process 17

5.1 Email Report Guidelines 17

6. Client-Developer Contractual Agreement 18

A. Appendices 19

A.1 Assembly Instructions to Implement 19

A.2 Diagrams 21

A.2.1 Frontend Diagram 21

A.2.2 Backend Diagram 21

A.3 Prototype Graphical User Interface Screen Capture 22

A.4 Features for Future Consideration 27

# 1. Introduction

The introduction to the Software Requirement Specification (SRS) document should provide an overview of the complete SRS document. While writing this document please remember that this document should contain all of the information needed by a software engineer to adequately design and implement the software product described by the requirements listed in this document. (Note: the following subsection annotates are largely taken from the IEEE Guide to SRS).

## 1.1 Purpose

The ASSIST/UNA software is primarily designed to enable students to learn and practice the ASSIST/I assembly language for the IBM/360. The primary users of this software will be students and instructors of CS 310 at the University of North Alabama. Students and instructors at other universities will also have access to the ASSIST/UNA software.

## 1.2 Scope

The ASSIST/UNA software is a Windows-based emulator of the ASSIST/I assembly language that will enable students to write assembly programs in a graphical environment. The ASSIST/UNA software will provide students with a subset of the functionality of the ASSIST/I assembler (see Sections 2 and 3). At this time, the ASSIST/UNA software is not meant to *fully* emulate all of the features of the ASSIST/I assembler; however, these features may be implemented later (see Appendix A.4).

The ASSIST/UNA software will be used by students and instructors in future CS 310 courses. This port to the Windows environment eliminates the need to use other emulating software, such as DOSBox, to write assembly programs. Also, the ASSIST/UNA IDE is designed to be user-friendly and to serve as a collection of useful tools, such as printing program reports. The ASSIST/UNA software will enable students to interact with an emulated ASSIST/I assembler in an intuitive and user-friendly environment.

## 1.3 Definitions, Acronyms, and Abbreviations

This section provides definitions of all terms, acronyms, and abbreviations required to properly interpret this software requirements specification document.

**1.3.1** **Assemble:** To translate source statements into corresponding machine code in the form of an object code program.

**1.3.2 Assembler:** A piece of software that assembles assembly source statements.

**1.3.3 ASSIST:** The Assembler System for Student Instruction and Systems Teaching for the IBM/360 (and IBM/370) developed by John Mashey and his students at Pennsylvania State University in the 1970s.

**1.3.4 ASSIST/I:** The version of ASSIST for personal computers.

**1.3.5 ASSIST/UNA:** The University of North Alabama emulator of the ASSIST/I assembler.

**1.3.6** **Client:** The person that has instigated development and has decided the project requirements.

**1.3.7 CS 310:** The Computer Science 310: Computer Organization & Assembly Language Programming course taught at the University of North Alabama.

**1.3.8 Developer:** The person, or persons, who actively design, implement, and maintain the ASSIST/UNA software and the accompanying documentation.

**1.3.9 DOSBox:** A Windows program that emulates the Windows DOS environment.

**1.3.10 Emulator:** Software on a hardware system that imitates the functionality of another software on its respective hardware system.

**1.3.11 GitHub:** A repository service (www.github.com) used to store remote copies of project source code and documentation.

**1.3.12 GUI:** Graphical User Interface.

**1.3.13 IBM:** International Business Machines.

**1.3.14 IBM/360:** A family of mainframes designed by IBM.

**1.3.15 IDE:** Integrated Development Environment.

**1.3.16 .NET Framework:** A programming framework developed by Microsoft. This project will be based on the .NET Framework version 4.5.

**1.3.17 .PRT File:** A program report file created by ASSIST/I or the ASSIST/UNA software.

**1.3.18 SRS:** Software Requirements Specification document.

**1.3.19 Team Foundation Server:** A Microsoft Visual Studio version-control system used to maintain and ensure the integrity of project source code.

**1.3.20 TFS:** Team Foundation Server.

**1.3.21 UNA:** The University of North Alabama in Florence, Alabama.

**1.3.22 User:** The person, or persons, who interact directly with the ASSIST/UNA software.

**1.3.23 Windows:** The Microsoft Windows operating system environment.

## 1.4 References

This software requirements specification document shall be used in conjunction with the following publications and client handouts.

Client, Assembly Instructions to Implement.[[1]](#footnote-1)

Client, Initial Project Description.[[2]](#footnote-2)

Client, More Detailed Description.[[3]](#footnote-3)

IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.[[4]](#footnote-4)

Microsoft, .NET Framework System Requirements.[[5]](#footnote-5)

## 1.5 Overview

The remainder of this software requirements document contains the high-level and detailed fundamental requirements of the ASSIST/UNA software. Section 2 provides a general, high-level understanding of the ASSIST/UNA software’s purpose, function, and user characteristics. Section 3 details the fundamental ASSIST/UNA software requirements, including functional requirements, use cases, and class/object relationships.

# 2. General Description

*This section of the SRS should describe the general factors that affect 'the product and its requirements. It should be made clear that this section does not state specific requirements; it only makes those requirements easier to understand.*

## 2.1 Product Perspective

The ASSIST/UNA software is a standalone product. Users of the software will be able to use it without requiring or relying on outside software.

## 2.2 Product Functions

The main function of the ASSIST/UNA software is to emulate the ASSIST/I assembler with a subset of the ASSIST/I’s instructions. The ASSIST/UNA software will enable users to write and assemble programs from an IDE. What is more, users will be able to assemble and debug their programs from the ASSIST/UNA environment. Users will also be able to view and print reports and source code directly from the IDE interface.

## 2.3 User Characteristics

The ASSIST/UNA software is to be used by the students and instructor of the CS 310 course at UNA, or in similar courses at other universities. Student users should have a basic understanding of programming fundamentals and basic understanding of the ASSIST/I assembly language. Although not required, student users with experience with IDEs will be benefited. Student users lacking experience with IDEs will find the ASSIST/UNA IDE to be intuitive and user-friendly. Instructor users should have a strong understanding of the ASSIST/I assembly language. The ASSIST/UNA IDE will enable instructor users to easily open, examine, and test student users’ programs. Instructor users will also be able to write their own programs.

## 2.4 General Constraints

The ASSIST/UNA software will need to reliably emulate the ASSIST/I assembler, especially since ASSIST/UNA is a teaching tool. This means that processing by and output of ASSIST/UNA will need to be at least 95% accurate. Also, the developers will have to adapt to the constraints of the C# language and the .NET Framework. Many visual components, such as buttons, are available by default when using Visual Studio. However, the team will inevitably have to create at least one custom component, namely a heavily modified text field.

## 2.5 Assumptions and Dependencies

The ASSIST/UNA software is to be produced for the Microsoft Windows operating system. Since the developers will be using C# and the .NET Framework version 4.5, only Windows Vista, Windows 7, and Windows 8/8.1 will be supported. The ASSIST/UNA software is being developed for the 64-bit versions of the Windows operating system.

# 3. Specific Requirements

This will be the largest and most important section of the SRS. The customer requirements will be embodied within Section 2, but this section will give the D-requirements that are used to guide the project’s software design, implementation, and testing.

Each requirement in this section should be:

* Correct
* Traceable (both forward and backward to prior/future artifacts)
* Unambiguous
* Verifiable (i.e., testable)
* Prioritized (with respect to importance and/or stability)
* Complete
* Consistent
* Uniquely identifiable (usually via numbering like 3.4.5.6)

Attention should be paid to the carefuly organize the requirements presented in this section so that they may easily accessed and understood. Furthermore, this SRS is not the software design document, therefore one should avoid the tendency to over-constrain (and therefore design) the software project within this SRS.

## 3.1 External Interface Requirements

This section details the user and software interfaces of the ASSIST/UNA software.

### 3.1.1 User Interfaces

3.1.1.1 Source Code Editor

The Source Code Editor provides a container in which users may write their ASSIST programs. Users may input data by typing into the Source Code Editor or by importing from another source file. The source program code in the Source Code Editor will serve as input to the ASSIST/UNA backend. The source program code in the Source Code Editor may also be saved to a file or printed. The Source Code Editor will automatically highlight the syntax of the source program code.

3.1.1.2 Registers Display

The Registers Display provides a container in which the assembler’s register contents may be displayed to the user. The registers within the Registers Display are read-only, and the registers do not serve as input to any module. The backend assembler will automatically update the registers within the Register Display. The registers will consist of eight hexadecimal characters. The Program Status Word, a special type of register, will consist of sixteen hexadecimal characters.

3.1.1.3 Memory Display

The Memory Display provides a container in which the assembler’s memory contents may be displayed to the user. The memory content within the Memory Display is read-only, and the memory content does not serve as input to any module. The backend assembler will automatically update the Memory Display. The Memory Display has the following format: Address; Memory Contents; and, EBCDIC Representation.

3.1.1.4 Symbol Table Display

The Symbol Table Display provides a container in which the assembler’s symbol table content may be displayed to the user. The symbol table content within the Symbol Table Display is read-only, and the symbol table content does not serve as input to any module. The backend assembler will automatically update the Symbol Table Display. The Symbol Table Display has the following format: Label; and, Address.

3.1.1.5 Output Dialog Window

The Output Dialog Window provides a container in which an assembled program’s output and errors may be displayed to the user. The program’s output and errors within the Output Dialog Window are read-only, and the program’s output and errors are also used to generate the .PRT file. The backend assembler will automatically update the Output Dialog Window.

### 3.1.2 Software Interfaces

3.1.2.1 Open File Dialog

The Open File Dialog enables the user to open source files for editing from within the ASSIST/UNA IDE. The user may open the Open File Dialog by selecting the corresponding menu option or toolbar option. The contents of the file selected within the Open File Dialog will be imported to the Source Code Editor (see Section 3.1.1.1). The Open File Dialog functionality is enabled by the Windows operating system.

3.1.2.2 Save File Dialog

The Save File Dialog enables the user to save source files from within the ASSIST/UNA IDE. The user may open the Save File Dialog by selecting the corresponding menu option or toolbar option. The contents of the Source Code Editor (see Section 3.1.1.1) will be saved to the file specified by the Save File Dialog. The Save File Dialog functionality is enabled by the Windows operating system.

3.1.2.3 Print Dialog

The Print Dialog enables the user print source program code or .PRT files from within the ASSIST/UNA IDE. The user may open the Print Dialog by selecting the respective menu option or toolbar option. The contents of the Source Code Editor (see Section 3.1.1.1) will be printed in portrait mode, or the contents of the .PRT file will be printed in landscape mode. The Print Dialog functionality is enabled by the Windows operating system.

## 3.2 Functional Requirements

This section details the fundamental functional requirements of the ASSIST/UNA software.

### 3.2.1 Source Code Editing

3.2.1.1 Introduction

The ASSIST/UNA software shall enable users to write and edit their assembly source programs within the IDE. Users will also have the ability to save their work or open previous work. The users will be able to print their source code from the IDE. Users shall also have access to basic functionality, such as copy, cut, and paste. In addition, users shall be able to perform search and search and replace queries.

3.2.1.2 Inputs

Users will either type their source program directly or open a file containing the source program code. Source program code loaded from files will be automatically imported into the text editor.

3.2.1.3 Processing

The text editor will apply syntax highlighting to the source program code. The exact syntax color scheme will be decided at a later date. Users will be able to assemble their source program code. In addition, users will have the option to debug assembled programs.

3.2.1.4 Outputs

Source program code may be saved to a file.

3.2.1.5 Error Handling

The user will be notified via message box if any text editor errors are encountered.

### 3.2.2 Source Code Assembly

3.2.2.1 Introduction

The ASSIST/UNA software shall enable users to assemble source program code from the IDE’s text editor.

3.2.2.2 Inputs

Input to the assembler shall be taken directly from the IDE’s text editor.

3.2.2.3 Processing

The backend system shall parse and process the source program code passed to the assembler. The assembler will update registers, memory content, and the symbol table as the source program code is assembled.

3.2.2.4 Outputs

Upon successful assembly of the source program code, the assembler shall generate and save a program report (.PRT file).

3.2.2.5 Error Handling

Errors encountered during assembly shall be displayed to the user via the output dialog window and reported in the .PRT file.

### 3.2.3 Source Code Assembly and Debugging

3.2.3.1 Introduction

The ASSIST/UNA software shall enable users to assemble source program code from the IDE’s text editor and to debug the assembled program.

3.2.3.2 Inputs

Input to the assembler shall be taken directly from the IDE’s text editor.

3.2.3.3 Processing

The backend system shall parse and process the source program code passed to the assembler. The assembler will update registers, memory content, and the symbol table as the source program code is assembled and as program statements are executed. Users shall have the ability to execute one program statement at a time.

3.2.3.4 Outputs

The contents of registers, memory, and the symbol table will be updated and displayed to the user (via the appropriate IDE components) after a program statement is executed. Any output from the program (e.g., XPRNT) shall be displayed in the output dialog window. A report file (.PRT file) will not be saved.

3.2.3.5 Error Handling

Errors encountered during assembly shall be displayed to the user via the output dialog window.

### 3.2.4 Source Code Assembly and Final Run

3.2.4.1 Introduction

The ASSIST/UNA software shall enable users to assemble source program code from the IDE’s text editor and to perform a final run.

3.2.4.2 Inputs

Input to the assembler shall be taken directly from the IDE’s text editor.

3.2.4.3 Processing

The backend system shall parse and process the source program code passed to the assembler. The assembler will update registers, memory content, and the symbol table as the source program code is assembled. The assembler will execute the program in full after successful assembly.

3.2.4.4 Outputs

The contents of registers, memory, and the symbol table will be updated and displayed to the user (via the appropriate IDE components) during program execution. Upon successful code assembly and program termination, the assembler shall generate and save a program report (.PRT file). Any program output (e.g., XPRNT) shall be displayed to the user via the output dialog window.

3.2.4.5 Error Handling

Errors encountered during assembly shall be displayed to the user via the output dialog window and reported in the .PRT file.

### 3.2.5 Report Viewing and Printing

3.2.5.1 Introduction

Users shall be able to view and print .PRT files.

3.2.5.2 Inputs

Users shall select either the view or print option from the IDE interface.

3.2.5.3 Processing

The .PRT file contents, if the file exists and is not empty, shall be displayed to the user in portrait mode in a separate viewing window. The .PRT file shall be printed in landscape mode. The Windows operating system shall handle the printing process.

3.2.5.4 Outputs

The .PRT file shall be displayed in a window or printed.

3.2.5.5 Error Handling

Errors encountered during assembly shall be displayed to the user via the output dialog window.

## 3.3 Use Cases

This section details the fundamental use cases of the ASSIST/UNA software.

### 3.3.1 Create a New Project

3.3.1.1 Description

The user creates a new working project to develop an assembly program.

3.3.1.2 Postconditions

The ASSIST/UNA IDE is reset to enable editing of a new, blank program.

3.3.1.3 Basic Flow

1. The user selects the option to create a new project.

2. The IDE text editor is cleared.

3. The registers are reinitialized to the F4F4F4F4 default value.

4. The memory content is reinitialized to the F5F5F5F5 default value.

5. The symbol table is cleared.

6. The user writes and edits the new source program code.

### 3.3.2 Save a Project

3.3.2.1 Description

The user saves a working project to a source file within the Windows file system.

3.3.2.2 Postconditions

The source program code is saved to a user-named file.

3.3.2.3 Basic Flow

1. The user selects the option to save a project.

2. The user is prompted to name the source file and the save location.

3. The contents of the IDE text editor are saved to the user-named source file.

### 3.3.3 Open a Project

3.3.3.1 Description

The user opens a project to edit.

3.3.3.2 Postconditions

The ASSIST/UNA IDE is reset and the opened source file’s code is loaded into the text editor.

3.3.3.3 Basic Flow

1. The user selects the option to open a project.

2. The user specifies the source file to be opened.

3. The contents of the source file are imported to the IDE text editor.

4. The registers are reinitialized to the F4F4F4F4 default value.

5. The memory content is reinitialized to the F5F5F5F5 default value.

6. The symbol table is cleared.

7. The user writes and edits the source program code.

### 3.3.4 Assemble a Program

3.3.4.1 Description

The user assembles the source program code.

3.3.4.2 Postconditions

The assembled program’s object code is saved. A .PRT file is saved. Errors encountered are reported in the .PRT file and in the output dialog window.

3.3.4.3 Basic Flow

1. The user selects the option to assemble the program source code.

2. The backend assembler assembles the program source code.

3. The assembled program’s object code is saved.

4. A .PRT file is saved.

5. Errors are reported in the .PRT file and in the output dialog window.

### 3.3.5 Assemble a Program and Debug

3.3.5.1 Description

The user assembles and debugs the source program code.

3.3.5.2 Postconditions

The assembled program’s object code is saved. Errors encountered are reported in the output dialog window. No .PRT file is saved.

3.3.5.3 Basic Flow

1. The user selects the option to assemble and debug the program source code.

2. The backend assembler assembles the program source code.

3. The assembled program’s object code is saved.

4. Errors are reported in the output dialog window.

5. The user executes one program statement at a time until program termination.

6. Program output is displayed in the output dialog window.

### 3.3.6 Assemble a Program and Perform a Final Run

3.3.6.1 Description

The user assembles and the source program code and performs a final run.

3.3.6.2 Postconditions

The assembled program’s object code is saved. Errors encountered are reported in the .PRT file and the output dialog window. A. PRT file is saved.

3.3.6.3 Basic Flow

1. The user selects the option to assemble the program source code and perform a final run.

2. The user specifies the final run identifier (usually the user’s name).

3. The backend assembler assembles the program source code.

4. The assembled program’s object code is saved.

5. Errors are reported in the output dialog window.

6. The assembled program executes.

7. A. PRT file, including errors, is generated and saved.

8. Program output is displayed in the output dialog window.

### 3.3.7 View a .PRT File

3.3.7.1 Description

The user views a .PRT file for an assembled program.

3.3.7.2 Postconditions

The .PRT file is displayed in a viewing window.

3.3.7.3 Basic Flow

1. The user selects the option to view the .PRT file.

2. If the .PRT file exists, it is displayed in portrait mode in a viewing window.

3. If the .PRT file does not exist, the user is notified accordingly.

### 3.3.8 Print a .PRT File

3.3.8.1 Description

The user prints a .PRT file for an assembled program.

3.3.8.2 Postconditions

The printing of the .PRT file is handled by the operating system.

3.3.8.3 Basic Flow

1. The user selects the option to print the .PRT file.

2. If the .PRT file exists, it is printed in landscape mode (via the operating system).

3. If the .PRT file does not exist, the user is notified accordingly.

### 3.3.9 Print Source Program Code

3.3.9.1 Description

The user prints the source program code.

3.3.9.2 Postconditions

The printing of the source program code is handled by the operating system.

3.3.9.3 Basic Flow

1. The user selects the option to print the source program code.

2. The source program code is printed in portrait mode (via the operating system).

## 3.4 Classes / Objects

This section contains the high-level descriptions of the classes and objects in the ASSIST/UNA backend and frontend, respectively.

### 3.4.1 Backend Classes

3.4.1.1 “Linker” (driver)

The “Linker” class is the main backend driver that runs and coordinates the individual backend modules. It is responsible for interfacing with each backend module, in addition to the frontend.

3.4.1.2 Assembler

The Assembler class contains all the assembler-related data, such as register contents, memory contents, and the symbol table.

3.4.1.3 Parser

The Parser class is responsible for parsing and tokenizing the source program code passed from the frontend to the backend “Linker.” The Parser will interface with the Library class. The Parser will also notify the “Linker” if there is an error.

3.4.1.4 Library

The Library class is used to verify that a program instruction token (from the Parser) is a valid ASSIST instruction. If the instruction token is valid, then the Library class uses the Definitions class to translate the instruction as appropriate.

3.4.1.5 Definitions

The Definitions class contains the definitions for each ASSIST instruction.

3.4.1.6 Error Detection

The Error Detection class is responsible for reporting the appropriate error (from the “Linker”) to the user. The Error Detection class will contain a list of labeled ASSIST assembly errors.

### 3.4.2 Frontend Classes

3.4.2.1 Driver

The Driver class is the main frontend driver that runs and coordinates the individual frontend modules. It is responsible for interfacing with each frontend module, in addition to the backend.

3.4.2.2 Source Code Editor

The Source Code Editor contains the user’s program source code. The Source Code Editor will feature line numbers and column headings (1-80). In addition, the Source Code Editor will feature syntax highlighting. When the user wishes to assemble, the program source code will be transferred from the Source Code Editor, through the Driver, to the backend.

3.4.2.3 Registers Display

The Registers Display has fields for all 16 general purpose registers (numbered 0 through 15) and the Program Status Word (including Condition Code). The Registers Display will display the contents of the assembler’s registers (see Section 3.4.1.2).

3.4.2.4 Memory Display

The Memory Display displays the memory contents of the assembler. The Memory Display has the following format: Address; Memory Contents; and, EBCDIC Representation. The Memory Display will gather its data from the assembler’s memory contents (see Section 3.4.1.2).

3.4.2.5 Symbol Table Display

The Symbol Table Display displays the assembler’s symbol table. The Symbol Table Display has the following format: Label; and, Address. The Symbol Table Display will gather its data from the assembler’s symbol table (see Section 3.4.1.2).

3.4.2.6 Output Dialog Window

The Output Dialog Window will display output from the XPRNT and the XDUMP instructions, in addition to any errors encountered during assembly (see Section 3.4.1.6).

## 3.5 Non-Functional Requirements

Non-functional requirements may exist for the following attributes. Often these requirements must be achieved at a system-wide level rather than at a unit level. State the requirements in the following sections in measurable terms (e.g., 95% of transaction shall be processed in less than a second, system downtime may not exceed 1 minute per day, > 30 day MTBF value, etc).

### 3.5.1 Performance

### 3.5.2 Reliability

### 3.5.3 Availability

### 3.5.4 Security

### 3.5.5 Maintainability

### 3.5.6 Portability

## 3.6 Design Constraints

Test cases involving source program code cannot be published publicly to GitHub. One of the developers is enrolled in CS 310 and cannot be allowed an unfair advantage. As such, he will be delegated to a primary role in frontend interface testing. Since the .NET Framework version 4.5 is supported only on Windows Vista, Windows 7, and Windows 8/8.1, the ASSIST/UNA software will not necessarily be portable to other operating systems.

## 3.7 Other Requirements

Users shall be able to modify the assembler options found in the original ASSIST/I via an options menu. These options include: Saving the output; changing the maximum number of output lines; changing the maximum number of program instructions; changing the maximum number of program pages; and, changing the maximum program size (in bytes). Users also shall not be restricted to one working directory. That is, users shall be able to specify absolute and relative paths to data files for assembled programs.

# 4. Deliverables

*Details of deliverables*

# 5. Change Management Process

Before reporting a change or an error, developers must ensure they are consulting the most recent version of the software requirements specification document. If a change is required or an error is encountered, this software requirements specification document will be updated by following this update process. Developers shall notify the technical writer and the team leader via email. The email will detail the section(s) and line number(s) that need to be changed. In addition, the email will contain detailed descriptions of each change or error. The reporting developer shall also provide a detailed suggested correction. The technical writer will review the correction reports, approve the corrections, and update this document as necessary. The new version of this document will be uploaded to the GitHub server, and each developer will be notified via email.

## 5.1 Email Report Guidelines

If a developer needs to report changes or errors in the software requirements specification document, he shall follow these guidelines:

* The email will be sent to the technical writer and the team leader.
* The email subject will read: “CS 455 – Spec Doc Changes.”
* The email will specify the version of the software requirements specification document.
* The email will contain the following sections for each change or error:
  + Section Number
  + Line Number
  + Detailed description of the change or error
  + Detailed suggested correction
* Developers should send themselves a carbon copy for their own records.

Each section of the email should be appropriately labeled. For example, the “Section Number(s)” section should be labeled “**Section Number(s)**.” Note that each reported change or error will need its own set of the aforementioned sections (with appropriate labels).

# 6. Client-Developer Contractual Agreement

The client (Dr. Patricia L. Roden) is satisfied that this software requirements specification document meets the client’s requirements and needs as specified at this time. The developers (listed below) recognize the client’s right to request modifications of the project requirements. The client will ensure the modifications are necessary and will notify the developers at least one week in advance. The client hereby approves this software requirements specification document in its current form. The developers agree thusly to deliver the agreed upon deliverables on or before April 29, 2014.

The client agrees to fairly assess the work delivered by the developers, in accordance with the specified requirements within this document. Additionally, the client agrees to assign to each developer the appropriate grade mark for his individual and overall contributions to the project. While grading the developers’ work, the client agrees to consult each developer’s evaluations. The signatures below hereby bind the client and the developers to this contractual agreement.

*Dr. Patricia L. Roden, Client* Date

*Travis Hunt, Team Leader* Date

*Michael Beaver, Technical Writer* Date

*Andrew Hamilton, Software Quality Assurance*  Date

*Drew Aaron* Date

*Clay Boren* Date

*Chad Farley* Date

# A. Appendices

Supplemental tables, figures, and miscellaneous information may be found within this appendix. The first section contains a list of ASSIST/I instruction types and the list of required assembly instructions to be implemented. The second section contains initial high-level frontend and backend diagrams. The third section contains screen captures of the prototype. The final section contains features that may not be implemented due to time constraints but may be desired by the client at a later time.

## A.1 Assembly Instructions to Implement

This appendix contains the list of required ASSIST/I assembly instructions to be implemented in the ASSIST/UNA emulator. Table A.1.1 lists the types of instructions supported by ASSIST/UNA. Table A.1.2 lists each ASSIST/I instruction’s mnemonic, description, instruction type, and basic form.

Table A.1.1: ASSIST/I instruction types supported by ASSIST/UNA.

|  |  |
| --- | --- |
| **Instruction Type** | **Type Description** |
| RR | Register-Register. Operands are taken from registers, manipulated, and the result is stored into a register (e.g., AR 3,4). |
| RS | Register-Storage. First operand is in a register, and the other operand is in main storage. Memory contents are accessed by D(B) addressing (e.g., LM 4,6,12(13)). |
| RX | Register-Storage. First operand is in a register, and the other operand is in main storage. Memory contents are accessed by D(X,B) addressing (e.g., LA 11,10(3,9)). |
| SI | Storage-Immediate. First operand is in main storage, and the second operand is an eight-bit (one byte) *immediate* operand. Memory contents are accessed by D(B) addressing (e.g., CLI 0(20),C’a’). |
| SS | Storage-Storage. Both operands are in main storage. Memory contents are accessed using D(B) addressing (e.g., MVC 0(5,3),0(2)). |
| X\* | An instruction whose mnemonic is preceded by an “X” is a special macro instruction that actually comprises several instructions. Memory contents are accessed using D(X,B) addressing (e.g., XDECI 3,0(1)). |

Table A.1.2: ASSIST/I Instructions to be implemented.

|  |  |  |  |
| --- | --- | --- | --- |
| **Mnemonic** | **Description** | **Type** | **Basic Form** |
| A | Add | RX | A R1,D2(X2,B2) |
| AP | Add Packed | SS | AP D1(L1,B1),D2(L2,B2) |
| AR | Add Register | RR | AR R1,R2 |
| BAL | Branch and Link | RX | BAL R1,D2(X2,B2) |
| BALR | Branch and Link Register | RR | BALR R1,R2 |
| BC | Branch on Condition | RX | BC B’mask’,D2(X2,B2) |
| BCR | Branch on Condition Register | RR | BCR B’mask’,R2 |
| BCT | Branch on Count | RX | BCT R1,D2(X2,B2) |
| BCTR | Branch on Count Register | RR | BCTR R1,R2 |
| BXH | Branch on Index High | RS | BXH R1,R2,D3(B3) |
| BXLE | Branch on Index Low or Equal | RS | BXLE R1,R2,D3(B3) |
| C | Compare | RX | C R1,D2(X2,B2) |
| CLC | Compare Logical Characters | SS | CLC D1(L1,B1),D2(B2) |
| CLI | Compare Logical Immediate | SI | CLI D1(B1),I2 |
| CP | Compare Packed | SS | CP D1(L1,B1),D2(L2,B2) |
| CR | Compare Register | RR | CR R1,R2 |
| D | Divide | RX | D R1,D2(X2,B2) |
| DP | Divide Packed | SS | DP D1(L1,B1),D2(L2,B2) |
| DR | Divide Register | RR | DR R1,R2 |
| ED | Edit | SS | ED D1(L1,B1),D2(B2) |
| EDMK | Edit and Mark | SS | EDMK D1(L1,B1),D2(B2) |
| L | Load | RX | L R1,D2(X2,B2) |
| LA | Load Address | RX | LA R1,D2(X2,B2) |
| LM | Load Multiple | RS | LM R1,R2,D3(B3) |
| LR | Load Register | RR | LR R1,R2 |
| M | Multiply | RX | M R1,D2(X2,B2) |
| MP | Multiply Packed | SS | MP D1(L1,B1),D2(L2,B2) |
| MR | Multiply Register | RR | MR R­1,R2 |
| MVC | Move Characters | SS | MVC D1(L1,B1),D2(B2) |
| MVI | Move Immediate | SI | MVI D1(B1),I2 |
| N |  |  |  |
| NR |  |  |  |
| O |  |  |  |
| OR |  |  |  |
| PACK | Pack | SS | PACK D1(L1,B1),D2(L2,B2) |
| S | Subtract | RX | S R1,D2(X2,B2) |
| SP | Subtract Packed | SS | SP D1(L1,B1),D2(L2,B2) |
| SR | Subtract Register | RR | SR R1,R2 |
| ST | Store | RX | ST R1,D2(X2,B2) |
| STM | Store Multiple | RS | STM R1,R2,D3(B3) |
| UNPK | Unpack | SS | UNPK D1(L1,B1),D2(L2,B2) |
| XDECI | Convert Input to Decimal | X\* | XDECI R1,D2(X2,B2) |
| XDECO | Convert Output to Decimal | X\* | XDECO R1,D2(X2,B2) |
| XDUMP | Dump Memory or Registers | X\* |  |
| XPRNT | Print | X\* | XPRNT D1(X1,B1),L2 |
| XREAD | Read Input | X\* | XREAD D1(X1,B1),L2 |
| ZAP | Zero, Add Packed | SS | ZAP D1(L1,B1),D2(L2,B2) |

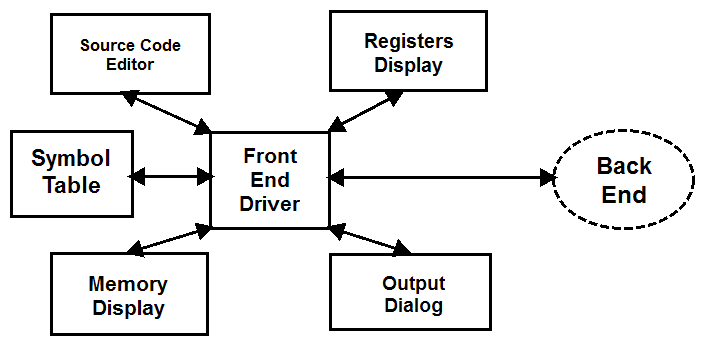
## A.2 Diagrams

This appendix contains the initial high-level frontend and backend diagrams, respectively. Note: These diagrams are *not* final and are subject to change.

### A.2.1 Frontend Diagram

This is the initial high-level frontend diagram. It is not a final design, and is meant to serve as a point of discussion and for visualization. All frontend modules will interact with the backend via a frontend driver. Bidirectional arrows indicate that information flows back and forth between modules.

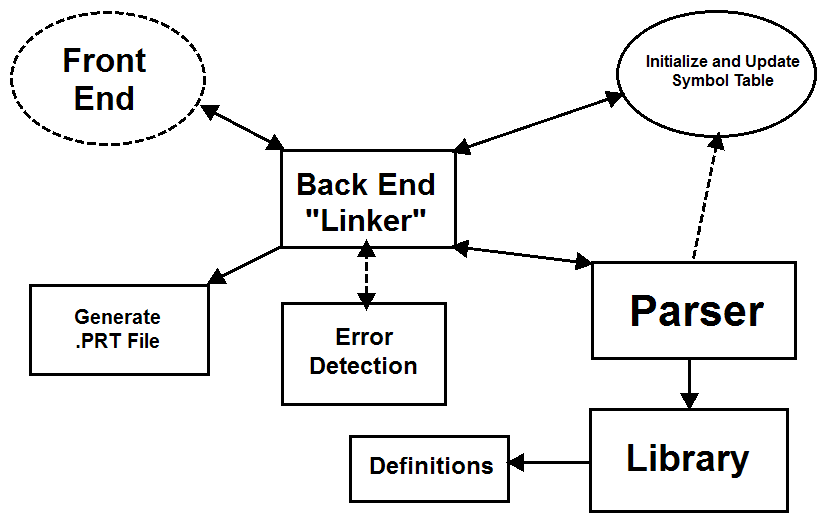
Figure A.2.1: The high-level frontend diagram.



### A.2.2 Backend Diagram

This is the initial high-level backend diagram. It is not a final design, and is meant to serve as a point of discussion and for visualization. All backend modules will interact with each other and the frontend via a backend driver, or “linker.”[[6]](#footnote-6) Directional arrows indicate the flow of information between modules. Bidirectional arrows indicate that information flows back and forth between modules. Dashed arrows indicate an indirect interaction.

Figure A.2.2: The high-level frontend diagram.



## A.3 Prototype Graphical User Interface Screen Capture

This appendix contains screen captures of the initial ASSIST/UNA prototype. Note: This prototype is *not* final and is subject to change. It is also worth noting that not all required features in this software requirements specification document are in the initial ASSIST/UNA prototype. This prototype is meant to serve as inspiration and as a point of discussion.

Figure A.3.1: The default IDE state.

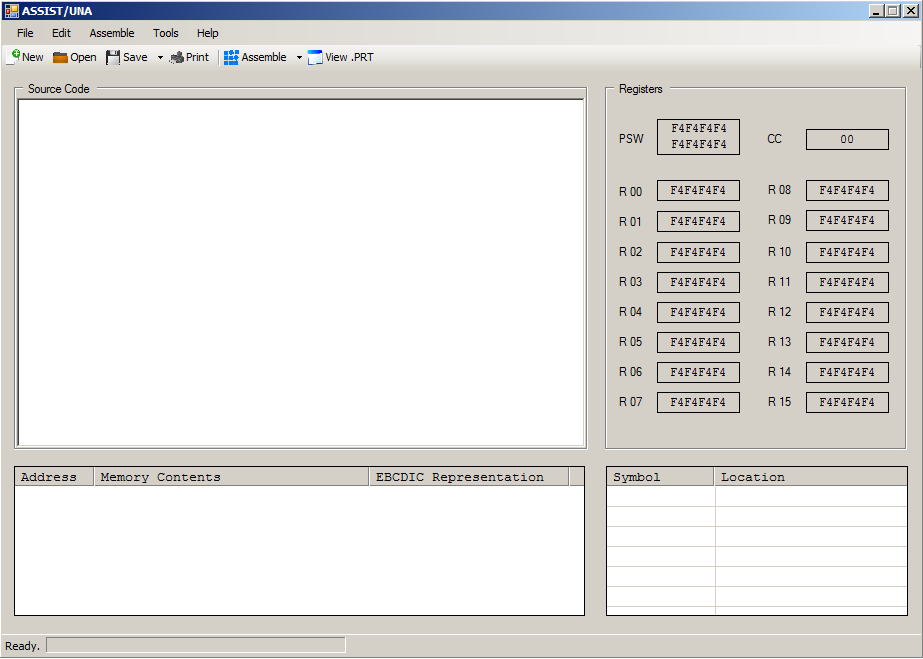


Figure A.3.2: Source program code in the IDE text editor. Note the highlighted comments.

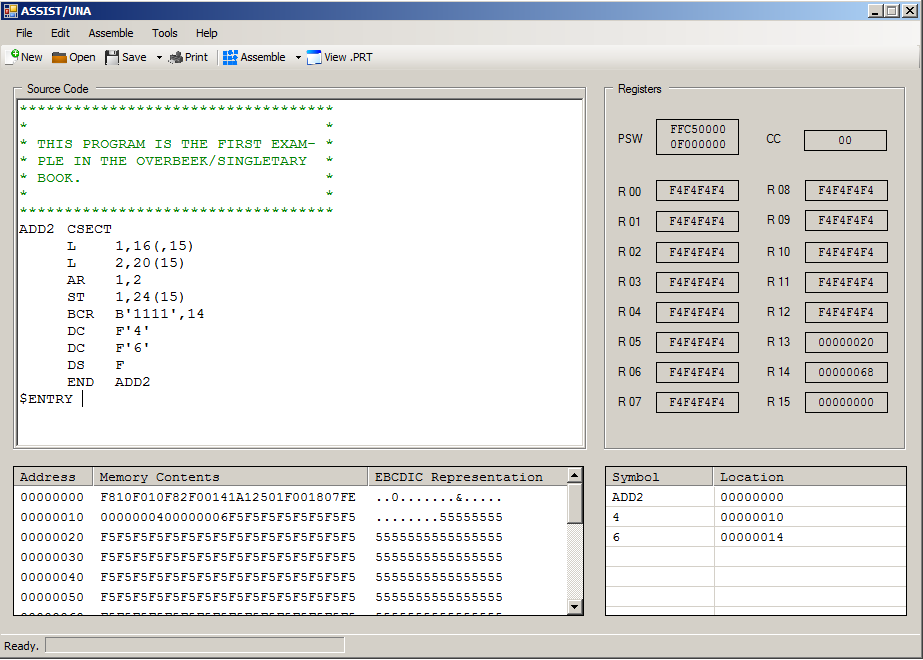


Figure A.3.3: The “File” menu options.

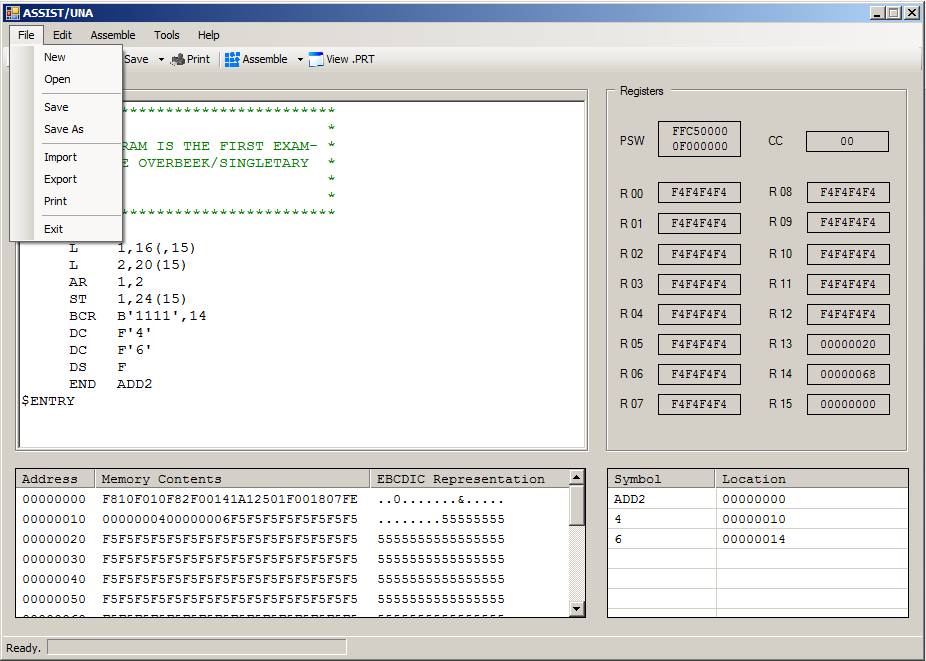


Figure A.3.4: The “Edit” menu options.

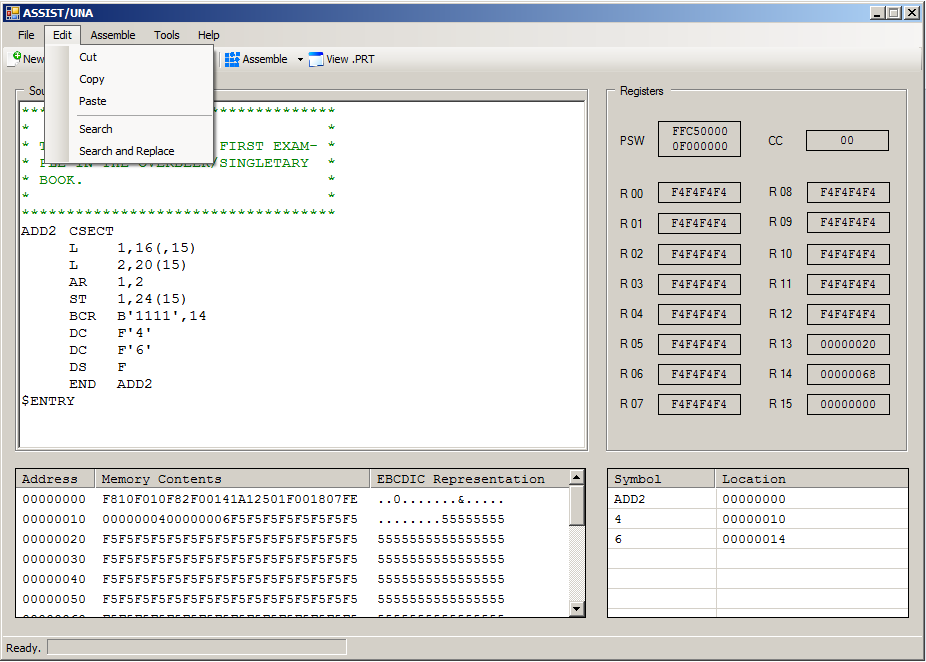


Figure A.3.5: The “Assemble” menu options.

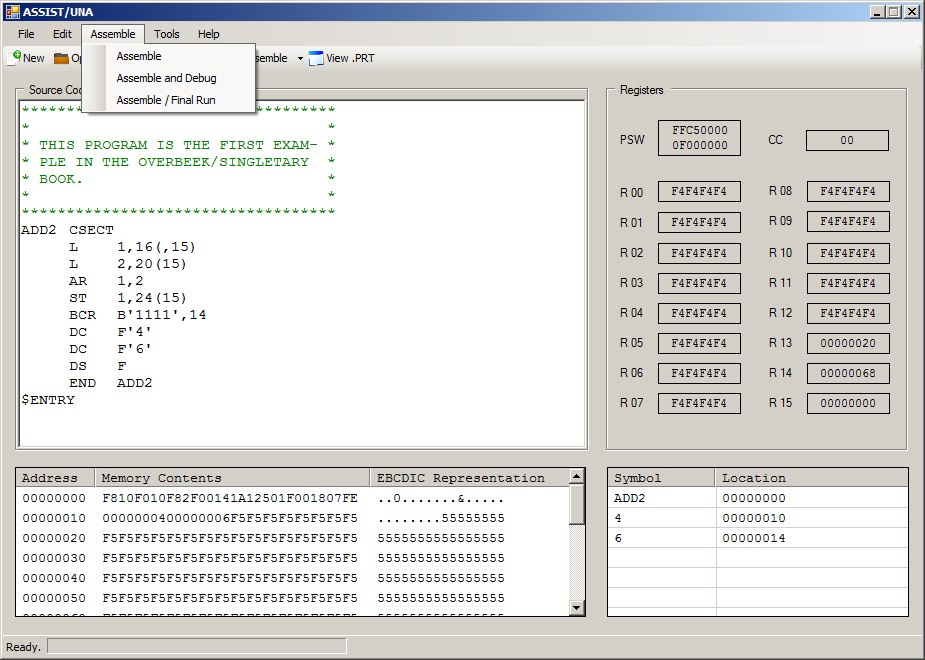


Figure A.3.6: The “Tools” menu options.

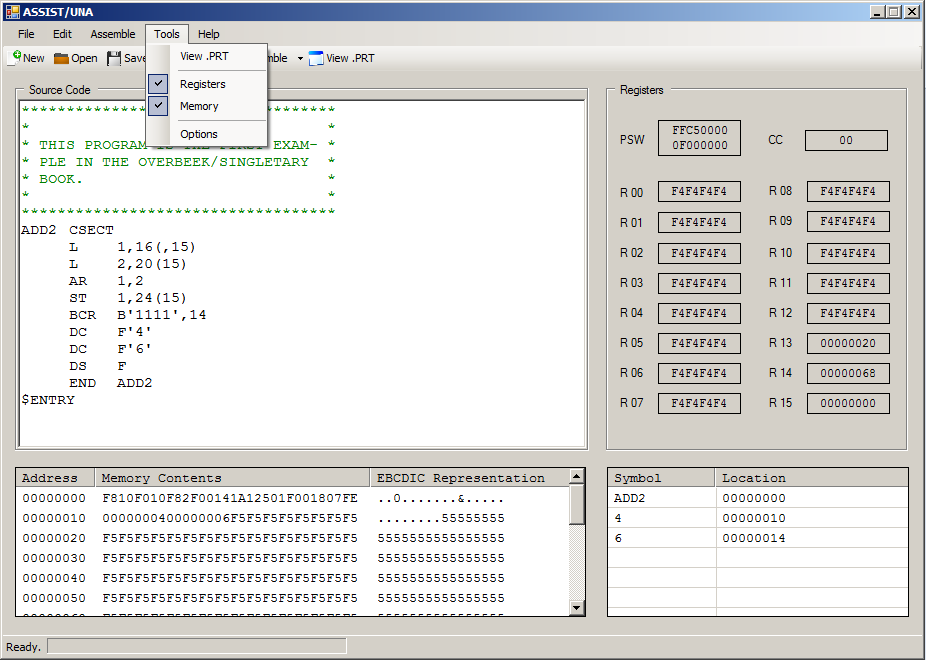


Figure A.3.7: The “Help” menu options.

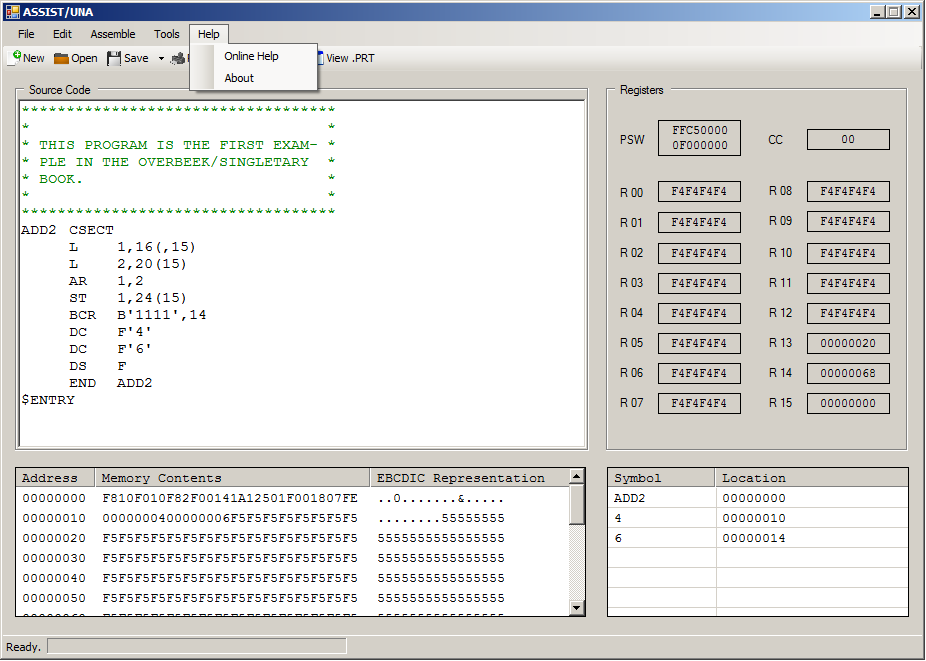


Figure A.3.8: The “Save” toolbar options.

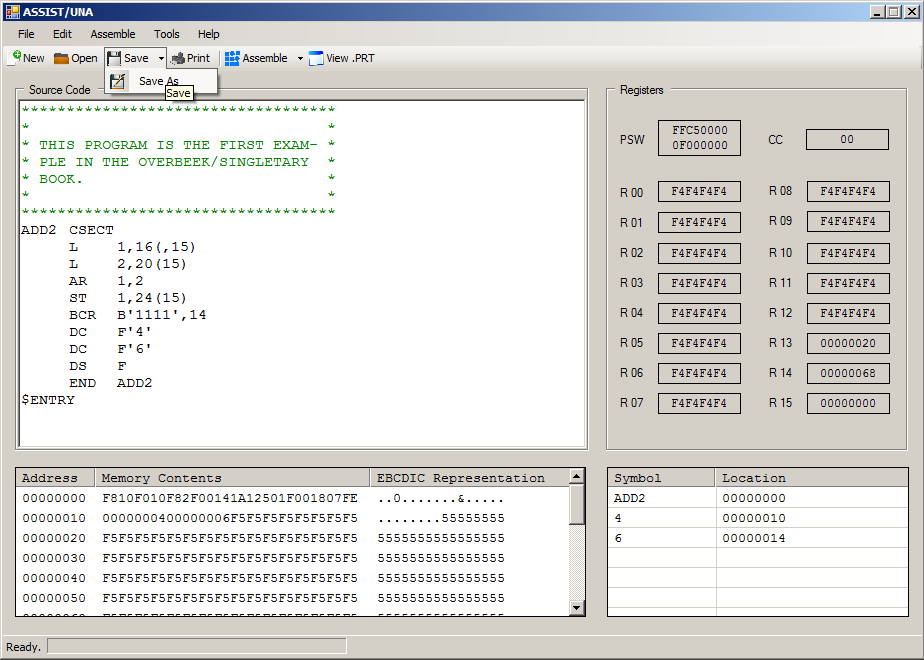
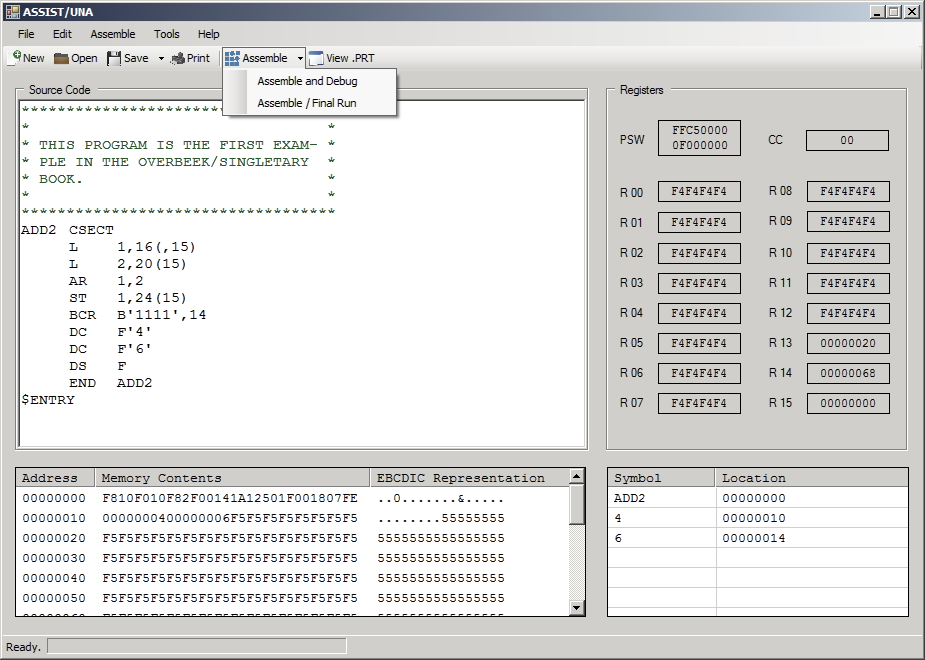


Figure A.3.9: The “Assemble” toolbar options.



## A.4 Features for Future Consideration

This appendix lists ASSIST/UNA software features that are not required but may be considered for implementation at a later date.

A.4.1 Formatted Tabs

When the user presses the tab key in the Source Code Editor (see Section 3.1.1.1), the cursor within the editor enters the appropriate amount of spaces.

A.4.2 Opcode Completion or Hinting

When the user begins entering an instruction’s opcode, the Source Code Editor (see Section 3.1.1.1) automatically completes the instruction opcode. Alternatively, the user could be presented with a list of matching instruction opcodes.

A.4.3 Embedded Calculator

The user presses a certain shortcut or hotkey to bring up a calculator within the IDE. The calculator should have the ability to calculate in and convert between hexadecimal, decimal, octal, and binary modes.

A.4.4 Disassembly

The user imports an assembled program that may be disassembled back into its original source program code.

A.4.5 Asterisk Reference

The user may use the asterisk within the source program code to refer to the location counter.

A.4.6 Floating-point Registers.

The ASSIST/UNA assembler should be able to emulate the IBM/360’s four floating-point registers.

1. Available on ANGEL. See Appendix A.1. [↑](#footnote-ref-1)
2. See footnote 1. See Sections 2 and 3. [↑](#footnote-ref-2)
3. See footnote 1. See Sections 2 and 3. [↑](#footnote-ref-3)
4. Available at http://standards.ieee.org/findstds/standard/830-1998.html. This IEEE guide is used throughout this software requirements specification document. [↑](#footnote-ref-4)
5. Available at http://msdn.microsoft.com/en-us/library/8z6watww%28v=vs.110%29.aspx. See Section 2.5. [↑](#footnote-ref-5)
6. The term “linker” is used because this module “links” together the other modules. [↑](#footnote-ref-6)