ASSIST/UNA
User Manual

Version 1.3
April 2014

Michael Beaver, *Technical Writer*
Clay Boren

University of North Alabama
CS 455: Software Engineering
Dr. Patricia L. Roden
Spring 2014

CONTENTS

1. Introduction

The ASSIST/UNA software is a Windows-based emulator of the ASSIST/I assembler for the IBM/360 system architecture. ASSIST/UNA features an internal assembler that mimics the ASSIST/I assembler and an internal simulator that emulates execution of assembled programs on an IBM/360 machine. This software is designed to be used by instructors teaching courses in assembly language programming and the students enrolled in those courses. Users should have at least some basic knowledge of programming principles and of the ASSIST assembly language for the IBM/360 architecture. The ASSIST/UNA environment is designed specifically to be more facilitative in assembly language programming than methods that have been previously employed, such as emulation via DOSBox. Fundamental information about the ASSIST/UNA software may be found in this user manual and in the accompanying online help manual.

2. The Graphical User Interface

The ASSIST/UNA graphical user interface (see Figure 1) is engineered to resemble modern interactive development environments, such as Microsoft's Visual Studio. As such, the main user interface incorporates all the components necessary for writing, assembling, and executing ASSIST assembly language programs. In this section, you will find descriptions, uses, and constraints of the various graphical components on the ASSIST/UNA user interface.

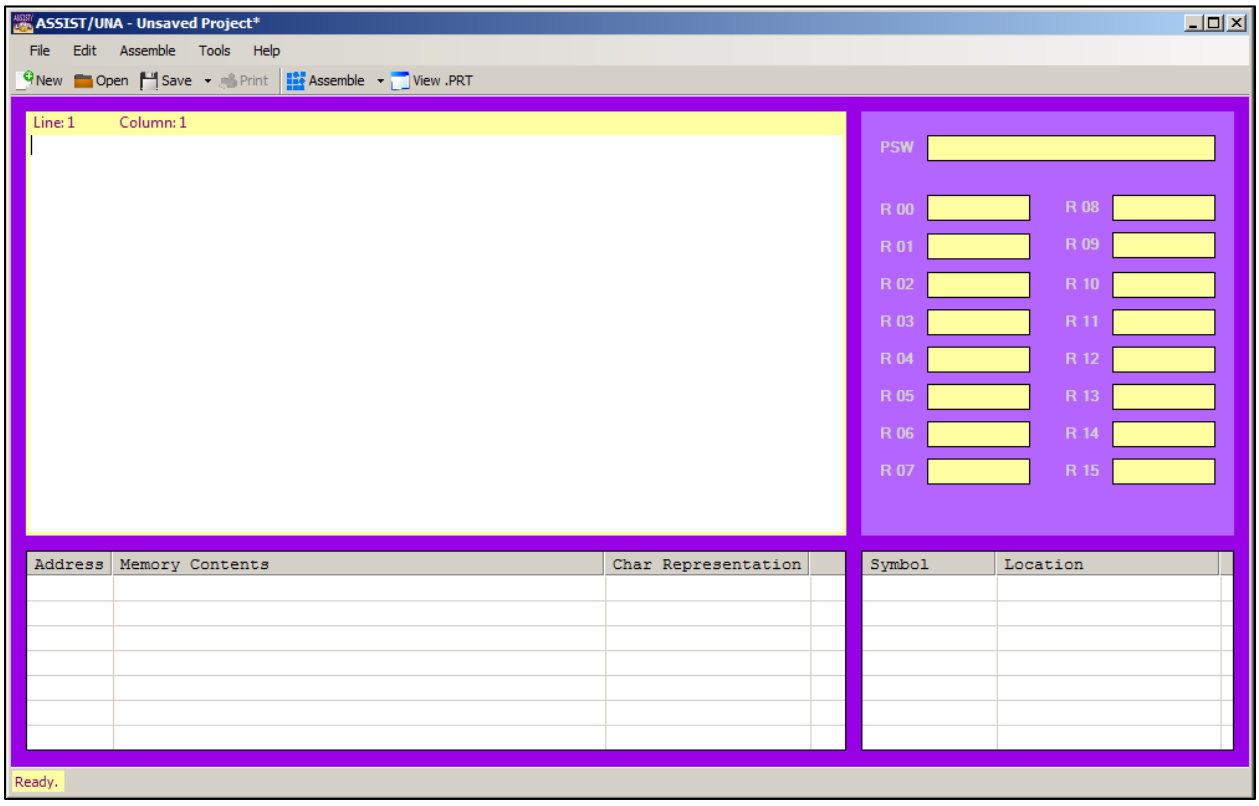

Figure 1: The ASSIST/UNA user interface in its default state.

2.1 The Menus and the Toolbar

ASSIST/UNA features a menu system and toolbar (see Figure 2) with some corresponding operations to facilitate development. This section includes descriptions and uses of the main menu system and the corresponding operations on the toolbar.

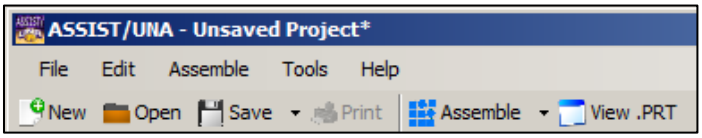

Figure 2: The ASSIST/UNA menu system and toolbar.

2.1.1 The File Menu

The File Menu (see Figure 3) is the first menu in the menu system, and it contains standard operations found in other environments.

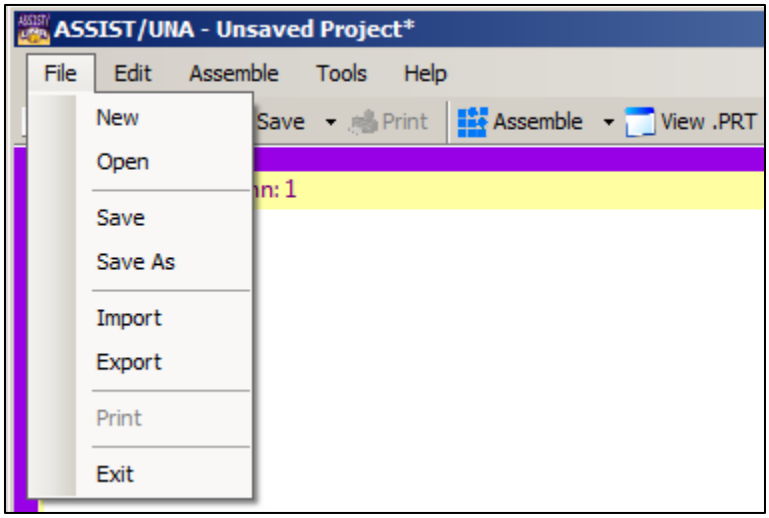

Figure 3: The File Menu.

The "New" (Ctrl+N) option resets the ASSIST/UNA environment to a default state. If the program is altered before selecting the "New" option, a prompt will appear asking whether or not the user wants to save changes. There is also a corresponding "New" button on the toolbar under the menu system. Clicking this button performs the same actions as going through the File Menu.

The "Open" (Ctrl+O) option prompts the user, via an open-file dialog, to open a .una file. All projects created in and saved by ASSIST/UNA take the form of .una files. Hence, only .una files may be selected using this option. If a program is already open and altered before the "Open" option is selected, the user will be prompted to indicate whether or not they want to save changes. Clicking the corresponding "Open" button on the toolbar performs the same actions as going through the File Menu.

The "Save" (Ctrl+S) and "Save As" options prompt the user, via a save-file dialog, to save their project as a .una file. Selecting the "Save" option will overwrite the original file, and the "Save As" option will allow the user to save the project to a new location or overwrite another existing

file. If the project to be saved is previously unsaved, the "Save" option behaves like the "Save As" option. The "Save" and "Save As" options on the toolbar are separated into a dropdown system. The "Save" button is the default option and behaves like the corresponding selection under the File Menu. By clicking on the dropdown arrow, the user has access to the "Save As" option, which behaves like the corresponding selection under the File Menu.

The "Import" option allows users to import, via an open-file dialog, the contents of plaintext file (.txt) and rich-text files (.rtf) into the ASSIST/UNA environment. This option is meant, primarily, to be used for importing programs originally developed in ASSIST/I. However, this option is also useful in conjunction with the "Export" option (see below) for editing data files. There is no corresponding button on the toolbar for this operation.

The "Export" option allows users to export, via a save-file dialog, the contents of the ASSIST/UNA source code editor (see Section 2.2) into a plaintext file (.txt) or a rich-text file (.rtf). This option is meant, primarily, to be used for exporting programs written in ASSIST/UNA so they may be opened in the ASSIST/I environment. However, this option is also useful in conjunction with the "Import" option (see above) for editing data files. There is no corresponding button on the toolbar for this operation.

The "Print" (Ctrl+P) option allows users to print their source code. The "Print" option is disabled until text is detected in the source code editor. The corresponding button on the toolbar performs the same actions as selecting the "Print" option under the File Menu. To print .PRT files, the user must first select the "View .PRT" option (see Section 2.7).

The "Exit" (Alt+F4) option closes the ASSIST/UNA environment. If a project is open and no changes have been made, the user will not be prompted to save. If changes are detected, the user will be prompted to indicate whether or not they wish to save changes upon exit.

2.1.2 The Edit Menu

The Edit Menu (see Figure 4) is the second menu in the menu system, and it contains standard operations found in other environments.

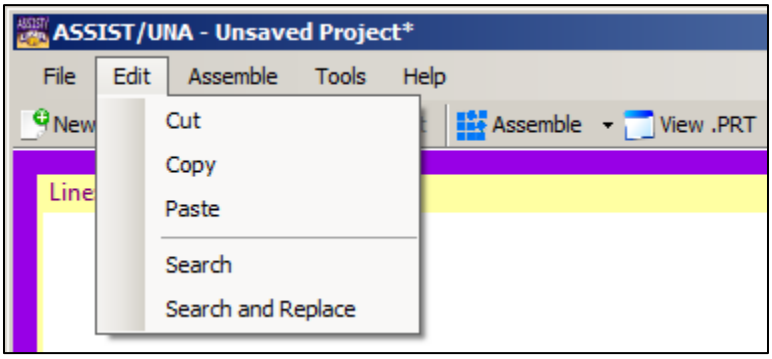

Figure 4: The Edit Menu.

The "Cut" (Ctrl+X), "Copy" (Ctrl+C), and "Paste" (Ctrl+V) options all perform their standard operations. There are no corresponding toolbar options for these operations.

The “Search” (Ctrl+F) option performs a search for a user-specified string within the source code editor. If the string is found, it will be highlighted. If the string is not found, nothing happens. There is no corresponding toolbar option for this operation.

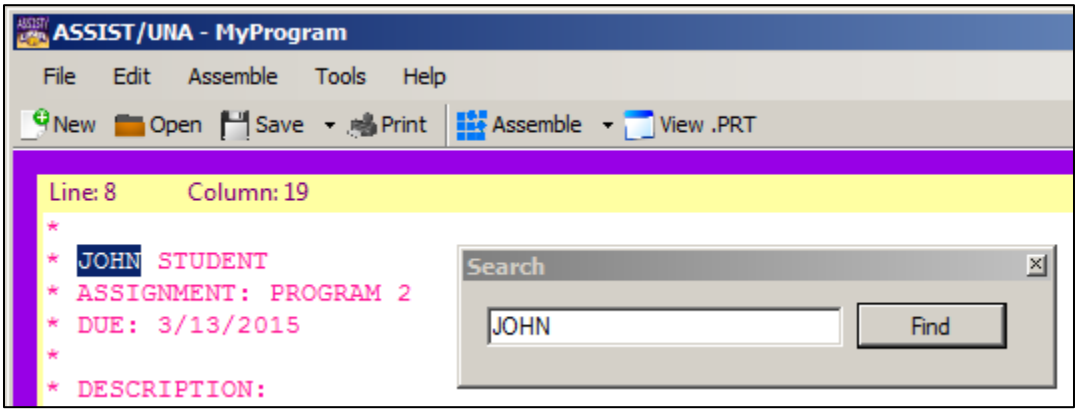

Figure 5: The “Search” operation in action.

The “Search and Replace” (Ctrl+H) option performs a search for a user-specified string within the source code editor. If the string is found, the user has the option to replace the found string with a user-specified replacement string. The replacement will only occur if the found string matches the original search string. There is no corresponding toolbar option for this operation.

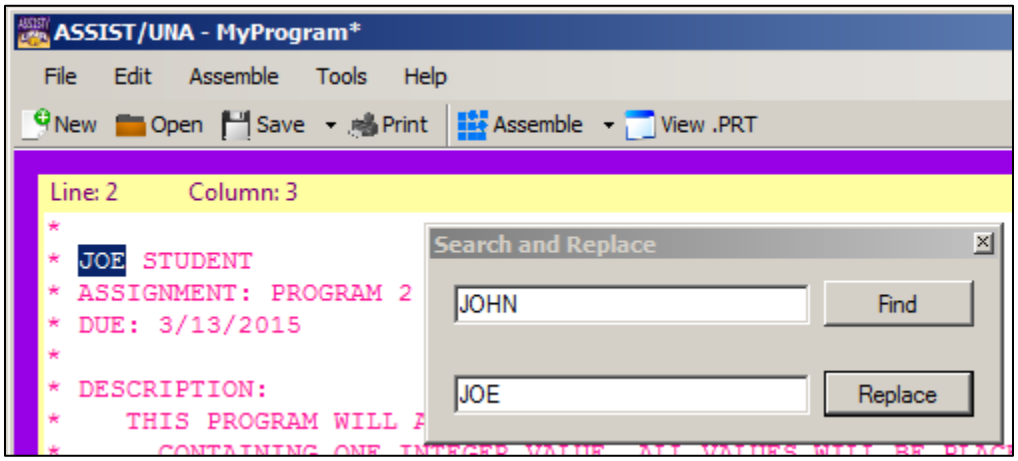

Figure 6: The “Search and Replace” operation in action.

2.1.3 The Assemble Menu

The Assemble Menu (see Figure 7) is the third menu in the menu system, and it contains operations specific to the ASSIST/UNA environment.

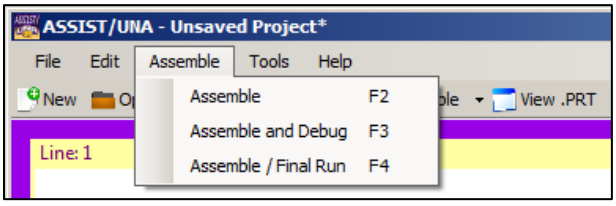

Figure 7: The Assemble Menu.

The “Assemble” (F2) option will attempt to assemble the contents of the source code editor into its appropriate object code. This option will generate and save a .PRT file, regardless of assembly success or failure. There is a corresponding option on the toolbar (see Figure 8). See Section 3 for more information.

The “Assemble and Debug” (F3) option will attempt to assemble the contents of the source code editor into its appropriate object code. This option will generate a .PRT file, but the file will not be saved. There is a corresponding option on the toolbar (see Figure 8). See Section 4.1 for more information.

The “Assemble / Final Run” (F4) option will attempt to assemble the contents of the source code editor into its appropriate object code. This option will generate and save a .PRT file, regardless of assembly success or failure. There is a corresponding option on the toolbar (see Figure 8). See Section 4.2 for more information.

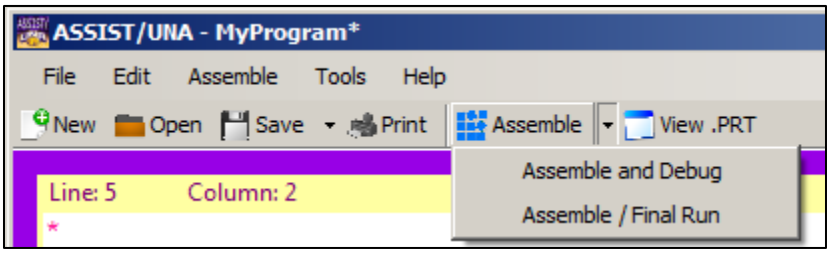

Figure 8: The Assemble toolbar options.

2.1.4 The Tools Menu

The Tools Menu (see Figure 9) is the fourth menu in the menu system, and it contains operations specific to the ASSIST/UNA environment.

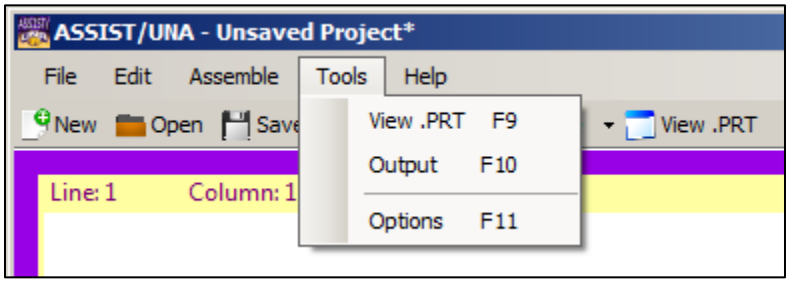

Figure 9: The Tools Menu.

The “View .PRT” (F9) option will open a new window to view the .PRT file associated with the project. The new window will not open if there is no associated .PRT file or if the file was deleted. There is a corresponding toolbar option for this operation. See Section 2.6 for more information.

The “Output” (F10) option will open a new window containing the output of the program. Exceptions encountered during execution and simulation will be reported in the new window. There is no corresponding toolbar option for this operation. See Section 2.7 for more information.

The “Options” (F11) option will open the Options window. Users are able to modify the options used in the assembler and the simulator. Users are also able to modify the appearance of the ASSIST/UNA environment. There is no corresponding toolbar for this operation. See Section 5 for more information.

2.1.5 The Help Menu

The Help Menu (see Figure 10) is the last menu in the menu system, and it contains ancillary operations.

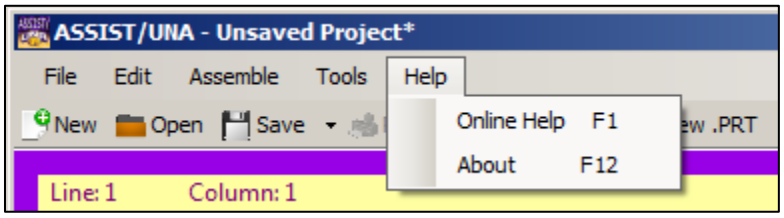

Figure 10: The Help Menu.

The “Online Help” (F1) option will open the accompanying online user manual. There is no corresponding toolbar option for this operation.

The “About” (F12) option will open the dialog window containing additional credits for the production of the ASSIST/UNA software. There is no corresponding toolbar for this operation.

2.2 The Source Code Editor

The source code editor (see Figure 11) provides an interface in which users may write their ASSIST programs. Program data may be either be typed directly, opened from another file, or imported from another file. The data in the source code editor may be printed (see Section 2.1.1). Comments (both asterisk-style and in-line style) will be highlighted in a color specified in the Options window (see Section 5). The source code editor also features ASSIST-like line numbers and column numbers. The source code editor is capable of basic functions, including copy, cut, and paste. Users may also perform search and search and replace operations.

All characters in the source code editor are forced to uppercase, as is done in the original ASSIST/I environment. Tab characters are replaced with spaces, and the tab key will move the caret to the appropriate ASSIST tab stop (i.e., column). The undo (Ctrl+Z) functionality has been disabled because it was neither efficient nor effective. Each row of text is limited to 79 characters. Text cannot be pasted over selected text; the selected text should be deleted first.

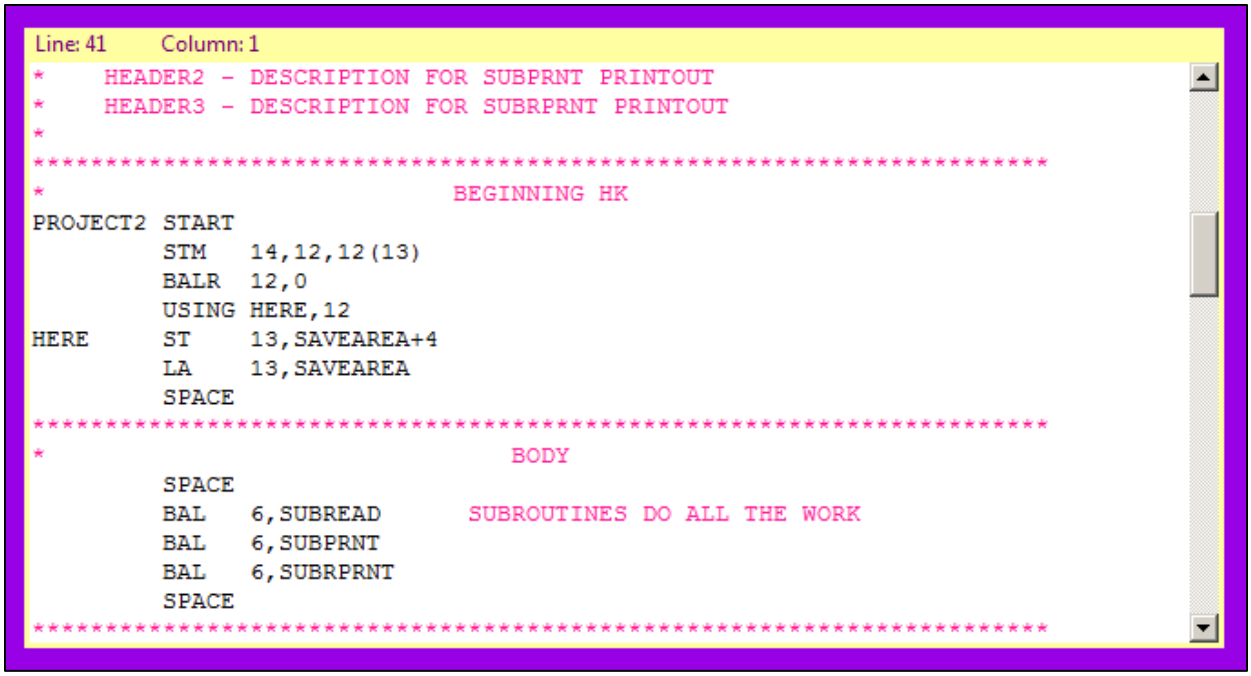

Figure 11: The source code editor with a program loaded.

2.3 The Registers Display

The registers display (see Figure 12) is located immediately to the right of the source code editor. The registers display features containers for the program status word (PSW) and for the sixteen general-purpose registers (R 00-15). The registers display maintains a default state of empty contents until debug or final run execution (see Section 4) occur. The contents are updated once execution begins and are not cleared until the ASSIST/UNA software is terminated or a new project is created. The contents are read-only, but their data may be copied by right-clicking and selecting the appropriate option.

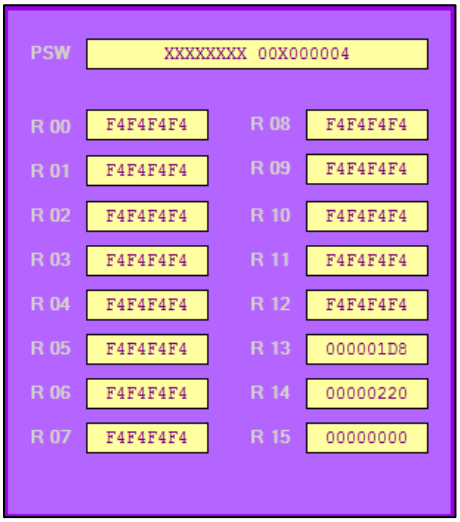

Figure 12: The registers display with register contents loaded.

The PSW container contains the data stored in the program status word. In the IBM/360 system, the program status word is 64 bits. However, for our purposes, we have masked ("X") the irrelevant parts of the PSW. After the separating space, the next two digits ("00" in Figure 12) represent the bits of the condition code. For example, the digits "10" would represent a condition code of 2 in the PSW. After the last masked digit, the remaining three bytes represent the instruction address. The instruction address is the location of the *next* instruction to be executed, not the current position of the location counter.

The general-purpose register containers represent the sixteen general-purpose registers of the IBM/360 system. These registers are updated during debug and final run executions.

2.4 The Memory Display

The memory display (see Figure 13) is located immediately below the source code editor. The memory display represents the main memory of the IBM/360 system. The default state is an empty container, and it is updated during debug or final run execution (see Section 4). The contents are updated once execution begins and are not cleared until the ASSIST/UNA software is terminated or a new project is created. The contents are read-only and cannot be copied.

The memory display is divided into three sections, and each row represents 16 (10_{16}) bytes in memory. The first section, "Address," is zero-based and reports the three-byte address locations (in hexadecimal) in memory. The second section, "Memory Contents," reports the hexadecimal representation of the bytes at each location, which are zero-based. Thus, the first entry in the row will be x0 and the final entry will be xF, where "x" is the address. Finally, the "Char Representation" section reports the printable characters (converted from EBCDIC). Characters that appear as periods—yet are not actual periods—represent nonprintable characters.

Address	Memory Contents	Char Representation
000000	90 EC D0 0C 05 C0 50 D0 C0 22 41 D0 C0 1E 45 60	..}..{.}{..}{..-
000010	C1 58 45 60 C1 7E 45 60 C1 A4 58 D0 C0 22 98 EC	A..-A=.-Au.}{.q.
000020	D0 0C 07 FE F5 F5 F5 F5 00 00 01 D8 F5 F5 F5 F5	}...5555...Q5555
000030	F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5	5555555555555555
000040	F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5	5555555555555555
000050	F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5	5555555555555555
000060	F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F0 40 40 40	5555555555550

Figure 13: The memory display with memory data loaded.

2.5 The Symbol Table Display

The symbol table display (see Figure 14) is located in the lower right corner of the ASSIST/UNA main user interface. The symbol table display represents the symbol table generated during assembly. The default state is an empty container, and it is updated during assembly, debug, and final run execution (see Section 4). The contents are not cleared until the ASSIST/UNA software is terminated or a new project is created. The contents are read-only and cannot be copied.

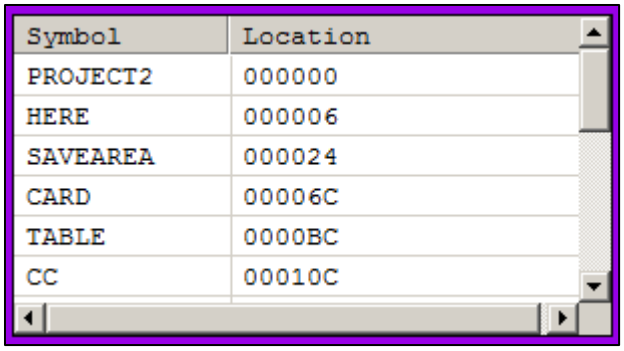

Figure 14: The symbol table with symbol data loaded.

The symbol table display is divided into two sections. The "Symbol" column reports the symbols (e.g., subroutine labels) encountered in the program during assembly. The "Location" column reports the locations in memory that each symbol may be found. The locations are given in hexadecimal and are three bytes in length.

2.6 The View .PRT Window

The View .PRT window (see Figure 15) is accessible via the Tools Menu and the toolbar (see Section 2.1.4). This window contains the .PRT file generated during assembly and after execution. The default state is an empty container, and it is updated during assembly, debug, and final run execution (see Section 4). The contents are cleared if the associated .PRT file is deleted or is inaccessible. The contents are read-only, but the contents may be copied.

If the associated .PRT file is nonexistent or is inaccessible, the View .PRT window will not open. Users may print .PRT files only from this window. This window may be accessed during debug but will be inaccessible once debug concludes. See Section 4.1 for more information.

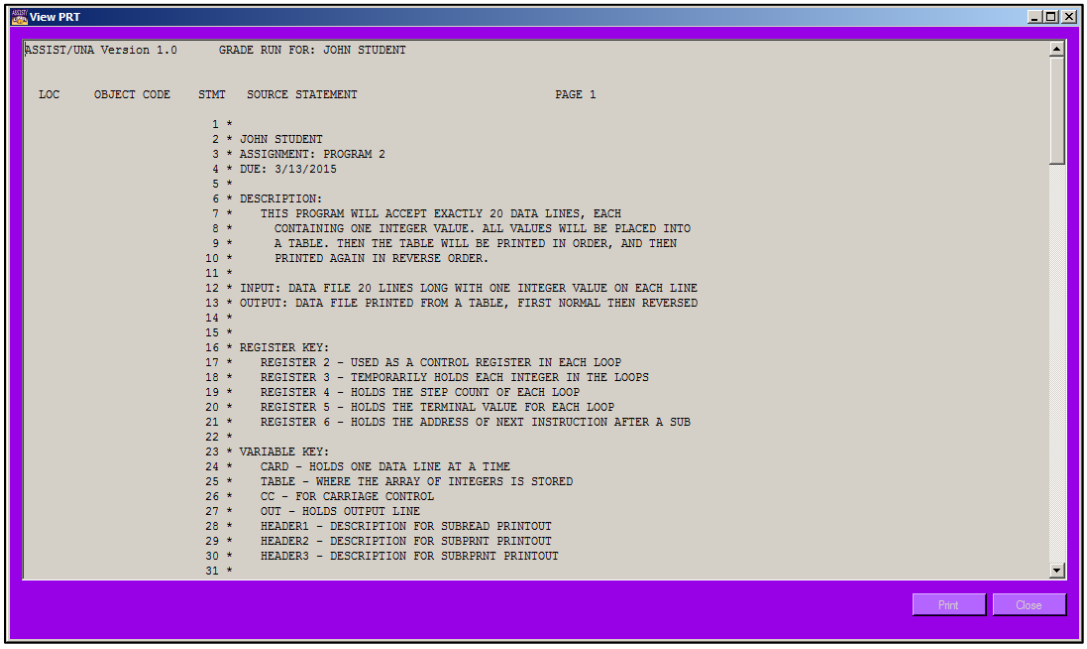

Figure 15: The View .PRT window with a sample .PRT file loaded.

2.7 The Output Window

The Output window (see Figures 16 and 17) is accessible via the Tools Menu (see Section 2.1.4). This window contains the program output and the results of a core dump if a fatal exception is encountered during execution and simulation. The default state is an empty container, and it is updated during debug and final run execution (see Section 4). The contents are cleared when a new simulation is started. The contents are read-only, but the contents may be copied.

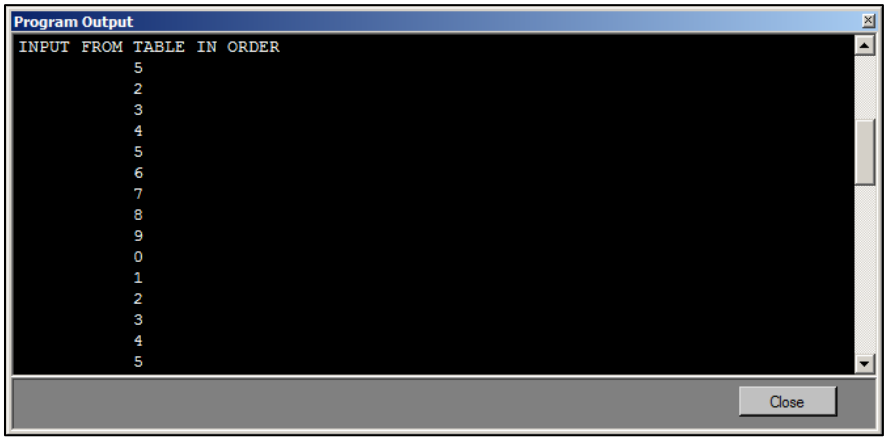

Figure 16: The Output window with sample program output loaded.

The Output window automatically displays after successful execution and simulation. The Output window also displays during debug mode with the option to hide it from view. This window will not automatically appear, however, if a fatal exception is experienced during execution and simulation. The Output window contents will be updated to contain the results of the core memory dump, but the window will not be automatically displayed.

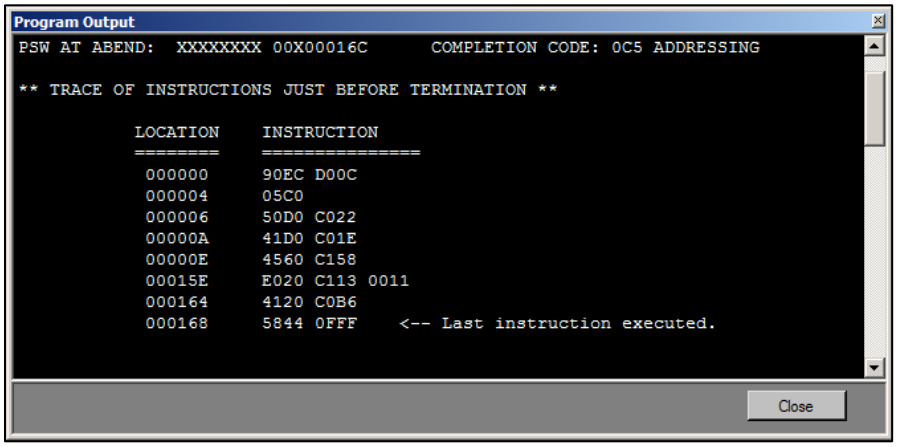

Figure 17: The Output window with sample core dump.

3. Program Assembly

The ASSIST/UNA environment is designed to emulate the ASSIST/I assembler. Hence, programs written in the ASSIST assembly language for the IBM/360 system may be assembled within ASSIST/UNA. A program may be assembled by selecting the "Assemble" option either via the Assemble Menu or the toolbar option (see Section 2.1.3). A program must be saved to an ASSIST/UNA project before it may be assembled.

Upon assembly, a message box will notify the user whether or not the program assembled with errors. Also, the symbol table display (see Section 2.5) will be updated to contain the contents of the internal symbol table generated during assembly. A .PRT file will also be generated by the assembler and will be available to view (see Section 2.6). Any errors encountered during assembly (see Appendix D) will be reported in the .PRT file. See Figure 18 for an example of the ASSIST/UNA environment after a program has been assembled.

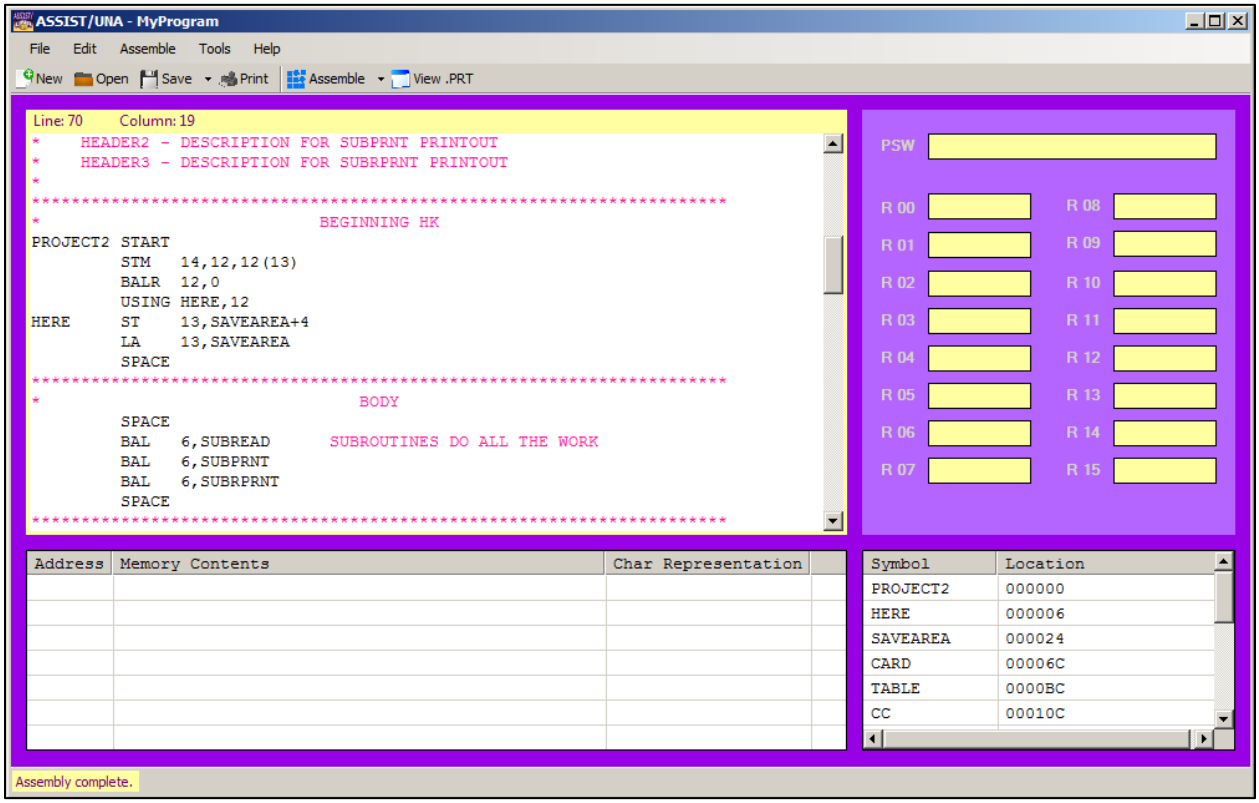

Figure 18: ASSIST/UNA after a program has been assembled.

The internal assembler operates under certain constraints. Source programs can have only up to 300 errors. The number of symbols, such as subroutine labels, in a program cannot exceed 100. The number of literals, such as =F'4', in a program cannot exceed 50 + 50*n*, where *n* is the number of uses of LTORG. For example, if a program uses LTORG twice, the maximum number of literals in that program would be 50 + 50*2 = 50 + 100 = 150 literals.

The internal assembler is also constrained by the values set in the Options window (see Section 5). The internal assembler will not assemble a file that exceeds the maximum number of source

lines as is specified in the Options. Similarly, the internal assembler will not generate a .PRT file if the number of pages within that .PRT file exceeds the maximum number of pages as specified in the Options.

4. Program Execution and Simulation

The ASSIST/UNA environment has two execution and simulation modes that emulate the execution of the program on an IBM/360 machine. The internal simulator is constrained by the settings in the Options menu (see Section 5). The internal simulator will only execute a finite number of instructions, as is specified in the Options. This protects the simulator from executing infinite loops. The internal simulator is also limited by the amount of memory it is allocated, as is specified in the Options. Hence, programs that exceed the specified memory size will not be executed.

4.1 Debug Mode

The debug mode may be accessed via the Assemble Menu or the corresponding toolbar option (see Figure 8). Debug mode will attempt to assemble the source program and execute the program line-by-line. Note that a .PRT file is temporarily saved and accessible via the View .PRT option during debug mode, but the file is deleted once debugging has ceased. All other non-debugger related menus and toolbar options are disabled during debug mode.

Upon activation of the debug mode, the Output window will appear with the option to hide it and reopen it later via the Tools Menu. Also, a new Debug Menu shall appear, and corresponding toolbar operations shall appear (see Figure 19). During debug, the memory display is maximized for better viewing, and the ASSIST/UNA window's resizing capability is disabled. The symbol table is automatically populated after the program is successfully assembled. After each instruction is executed, the memory display and the registers display are updated.

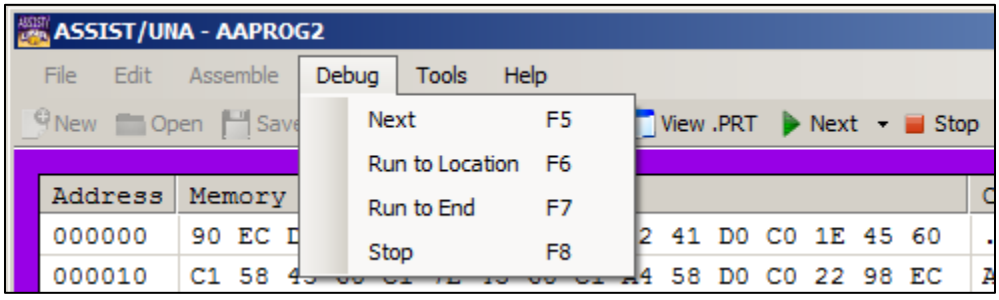

Figure 19: The Debug Menu.

4.1.1 The Debug Menu

The "Next" (F5) option executes the next instruction (i.e., the instruction to which the location counter is currently pointing). This option is accessible via the Debug Menu and the corresponding toolbar option (see Figure 20).

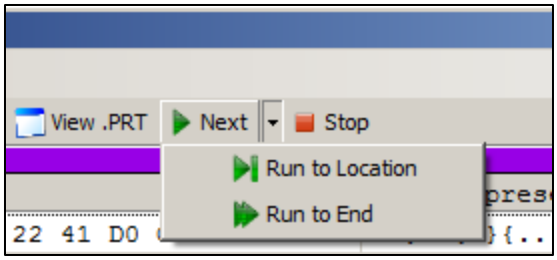

Figure 20: The corresponding toolbar debug operations.

The "Run to Location" (F6) option prompts the user to specify a three-byte address for the location counter to run until (see Figure 21). Like a breakpoint, the debugger will be paused once the location counter reaches the specified address. The debugger will not execute the instruction at that location until the user selects the "Next" operation. If the specified location is invalid (out of range or already passed), the debugger will run until the end of the program. This option may be accessed via the Debug Menu or the corresponding toolbar option.

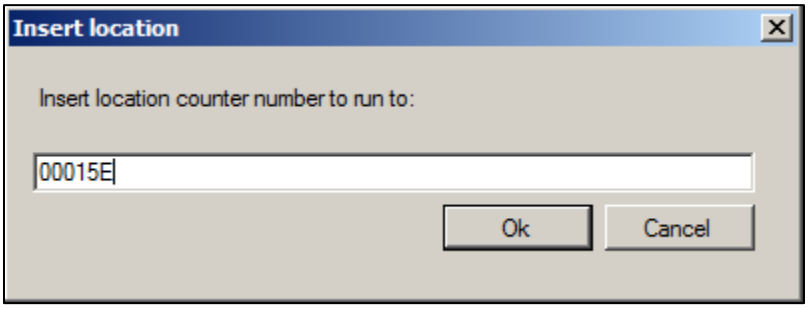

Figure 21: The location prompt.

The "Run to End" (F7) option completes execution of the program from the current position of the location counter. Choosing this option at the beginning of debug is similar to selecting the "Assemble/Final Run" option (see Section 4.2). This option is accessible via the Debug Menu or by the corresponding toolbar option.

The "Stop" (F8) option halts all debugging and execution of the program. This option is accessible via the Debug Menu or by the corresponding toolbar option.

4.2 Final Run

The final run mode may be accessed via the Assemble Menu or the corresponding toolbar option (see Figure 8). Final run mode will attempt to assemble the source program and execute the program without breaking. Note that a .PRT file is saved and accessible via the View .PRT window.

When the final run option is selected, the user is prompted to enter an identifier for the .PRT file (see Figure 22). The identifier can be blank, but best practice is to use the programmer's name. If the program executes successfully, the Output window will appear with the program's output. The program output will also be recorded in the .PRT file. However, if execution fails due to a fatal exception, the Output window will not appear. Instead, a core memory dump will be performed and saved to the .PRT file, along with any partial output that was successfully created.

Note that while the Output window does not appear in the case of a fatal exception, the core memory dump is still recorded in the Output window.

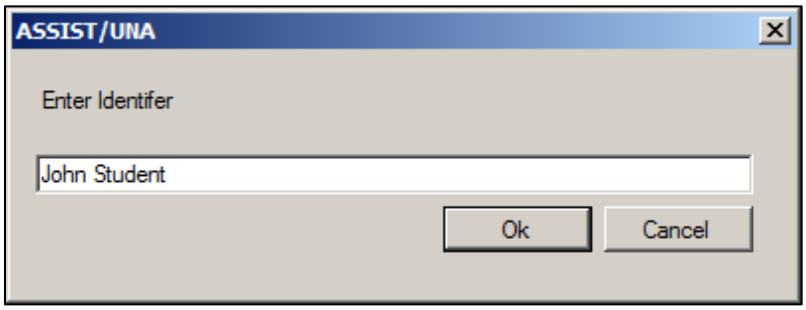

Figure 22: The identifier prompt.

The registers and memory displays are also updated during final run execution. Since execution is rather fast, the displays may not update immediately. The contents of the displays at the end of execution accurately reflect the final state of the simulated machine.

5. Options

The Options window (see Figure 23) allows the user to modify aspects of the ASSIST/I simulator and the ASSIST/UNA user interface. These options are grouped accordingly within the Options window. The Options window may be accessed via the Tools Menu (see Section 2.1.4).

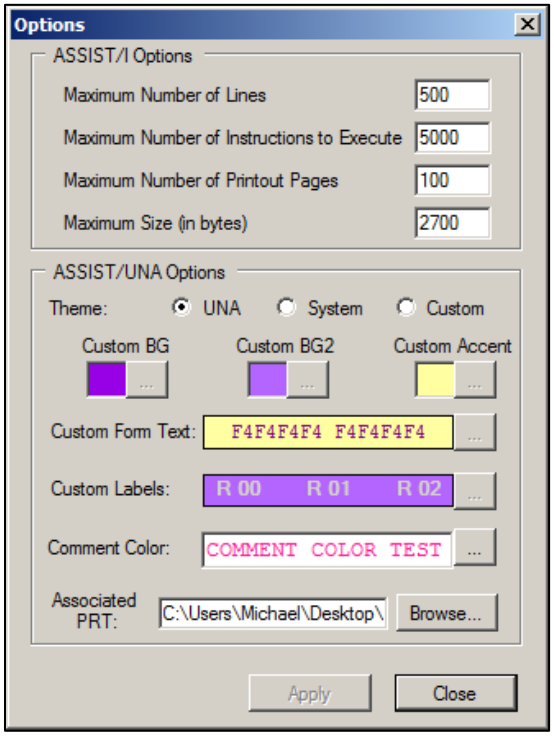

Figure 23: The Options window.

5.1 ASSIST/I Options

The "ASSIST/I Options" in the Options window correspond directly to the options in the original ASSIST/I environment. These options directly constrain the ASSIST/UNA internal assembler and simulator.

The "maximum number of lines" field sets the maximum number of lines that a .PRT file may contain. If this bound is exceeded, the internal assembler presents the user with an error message. The "maximum number of instructions to execute" field sets the maximum number of instructions that the internal simulator can execute. This bound is used primarily to prevent the internal simulator from falling into infinite loops. If this bound is exceeded, program execution and simulation are halted. The "maximum number of printout pages" field sets the maximum number of pages that the internal assembler can generate for the .PRT file. If this bound is exceeded, the assembler presents the user with an error message. The "maximum size" field sets maximum size of the internal simulator's memory (in bytes). If this bound is too low, it is possible that the program may not be loaded and executed. The default size is recommended, except possibly for exceptionally large programs.

5.2 ASSIST/UNA Options

The "ASSIST/UNA Options" enable the user to modify the appearance of the ASSIST/UNA environment and to associate a .PRT file with a project. The user has the option to use the default UNA color scheme, a system (grayscale) color scheme, or a specified color scheme.

All ASSIST/UNA projects have associated .PRT files. If there is not an associated .PRT file, the user will be notified and a file shall be automatically generated. A .PRT file for a project is associated with the project after executing any of the Assemble Menu operations. Users have the option to specify a different .PRT file if the currently associated file is incorrect, for example. The user does not have to manually associate a .PRT file with an ASSIST/UNA project.

Appendix A: System Requirements

ASSIST/UNA is built using the Microsoft .NET Framework 4.5. Hence, users that wish to use ASSIST/UNA must ensure their systems meet at least the following basic requirements.[1]

Table 1: Hardware Requirements

Processor	1 GHz
RAM	512 MB
Minimum Disk Space (32-bit)	850 MB
Minimum Disk Space (64-bit)	2 GB

Table 2: Supported Operating Systems

Windows 8.1	32-bit and 64-bit
Windows 8	32-bit and 64-bit
Windows 7 SP1	32-bit and 64-bit
Windows Vista SP2	32-bit and 64-bit

[1] This information may be retrieved at http://msdn.microsoft.com/en-us/library/8z6watww%28v=vs.110%29.aspx.

Appendix B: Supported ASSIST/I Instructions

ASSIST/UNA supports the instructions listed in Table 3. The START, END, TITLE, EJECT, SPACE, LTORG, CSECT, USING, DS, and DC assembler directives are also supported. The various extended mnemonics for BC and BCR are supported by ASSIST/UNA.

Table 3: Supported ASSIST/I instructions.

Mnemonic	Description	Type	Basic Form
A	Add	RX	A R_1,D_2(X_2,B_2)
AP	Add Packed	SS	AP D_1(L_1,B_1),D_2(L_2,B_2)
AR	Add Register	RR	AR R_1,R_2
BAL	Branch and Link	RX	BAL R_1,D_2(X_2,B_2)
BALR	Branch and Link Register	RR	BALR R_1,R_2
BC	Branch on Condition	RX	BC B'mask',D_2(X_2,B_2)
BCR	Branch on Condition Register	RR	BCR B'mask',R_2
BCT	Branch on Count	RX	BCT R_1,D_2(X_2,B_2)
BCTR	Branch on Count Register	RR	BCTR R_1,R_2
BXH	Branch on Index High	RS	BXH R_1,R_2,D_3(B_3)
BXLE	Branch on Index Low or Equal	RS	BXLE R_1,R_2,D_3(B_3)
C	Compare	RX	C R_1,D_2(X_2,B_2)
CLC	Compare Logical Characters	SS	CLC D_1(L_1,B_1),D_2(B_2)
CLI	Compare Logical Immediate	SI	CLI D_1(B_1),I_2
CP	Compare Packed	SS	CP D_1(L_1,B_1),D_2(L_2,B_2)
CR	Compare Register	RR	CR R_1,R_2
D	Divide	RX	D R_1,D_2(X_2,B_2)
DP	Divide Packed	SS	DP D_1(L_1,B_1),D_2(L_2,B_2)
DR	Divide Register	RR	DR R_1,R_2
ED	Edit	SS	ED D_1(L_1,B_1),D_2(B_2)
EDMK	Edit and Mark	SS	EDMK D_1(L_1,B_1),D_2(B_2)
L	Load	RX	L R_1,D_2(X_2,B_2)
LA	Load Address	RX	LA R_1,D_2(X_2,B_2)
LM	Load Multiple	RS	LM R_1,R_2,D_3(B_3)
LR	Load Register	RR	LR R_1,R_2
M	Multiply	RX	M R_1,D_2(X_2,B_2)
MP	Multiply Packed	SS	MP D_1(L_1,B_1),D_2(L_2,B_2)
MR	Multiply Register	RR	MR R_1,R_2
MVC	Move Characters	SS	MVC D_1(L_1,B_1),D_2(B_2)
MVI	Move Immediate	SI	MVI D_1(B_1),I_2
N	Bitwise AND	RX	N R_1,D_2(X_2,B_2)
NR	Bitwise AND Register	RR	NR R_1,R_2
O	Bitwise OR	RX	O R_1,D_2(X_2,B_2)
OR	Bitwise OR Register	RR	OR R_1,R_2
PACK	Pack	SS	PACK D_1(L_1,B_1),D_2(L_2,B_2)

S	Subtract	RX	S R_1,D_2(X_2,B_2)
SP	Subtract Packed	SS	SP D_1(L_1,B_1),D_2(L_2,B_2)
SR	Subtract Register	RR	SR R_1,R_2
ST	Store	RX	ST R_1,D_2(X_2,B_2)
STM	Store Multiple	RS	STM R_1,R_2,D_3(B_3)
UNPK	Unpack	SS	UNPK D_1(L_1,B_1),D_2(L_2,B_2)
XDECI	Convert Input to Decimal	X*	XDECI R_1,D_2(X_2,B_2)
XDECO	Convert Output to Decimal	X*	XDECO R_1,D_2(X_2,B_2)
XPRNT	Print Output	X*	XPRNT D_1(X_1,B_1),L_2
XREAD	Read Input	X*	XREAD D_1(X_1,B_1),L_2
ZAP	Zero, Add Packed	SS	ZAP D_1(L_1,B_1),D_2(L_2,B_2)

Appendix C: Assemble-Time Errors

This appendix contains basic descriptions of the types of errors that may be encountered during assembly. These errors are non-fatal and are reported in the .PRT file under the offending source statement.

AS005 END CARD MISSING-SUPPLIED
The assembler creates an END card because the user has supplied none before an end-file marker.

AS100 ADDRESSIBILITY ERROR
An implied address is used which cannot be resolved into base displacement form. No base register is available having the same relocatability attribute and a value from 0 to 4095 less than the value of the implied address.

AS102 ILLEGAL CONSTANT TYPE
An unrecognizable type of constant is specified.

AS103 CONTINUATION CARD COLS. 1-15 NONBLANK
A continuation card contains nonblank characters in columns 1-15. This may be caused by an accidental punch in column 72 of the preceding card.

AS106 TOO MANY OPERANDS IN DC
ASSIST allows no more than ten operands in a DC statement.

AS108 ILLEGAL DUPLICATION FACTOR
A duplication factor either exceeds the maximum value of 32,767, or a duplication factor in a literal constant is not specified by a decimal term or else has the value zero.

AS109 EXPRESSION TOO LARGE
The value of the flagged expression or term is too large for the given usage, such as a constant length greater than the maximum permissible for the type of constant.

AS112 LABEL NOT ALLOWED
A label is used on a statement not permitting one, such as a CNOP or USING statement.

AS114 INVALID CONSTANT
A constant contains invalid characters for its type, or is specified improperly in some other way.

AS115 INVALID DELIMITER
The character flagged cannot appear in the statement where it does. This message is used whenever the scanner expects a certain kind of delimiter to be used, and it is not there.

AS117 INVALID SYMBOL
The symbol flagged either contains nine or more characters or does not begin with an alphabetic character as is required.

AS118 INVALID OP-CODE
The statement contains an unrecognizable mnemonic op-code, or none at all. Note that different versions of ASSIST may not accept some of the possible op-codes.

AS119 PREVIOUSLY DEFINED SYMBOL
The symbol in the label field has been previously used as a label, or a SET variable has been previously declared.

AS120 ABSOLUTE EXPRESSION REQUIRED
A relocatable expression is used where an absolute one is required, such as in constant duplication factor or for a register.

AS121 MISSING DELIMITER
A delimiter is expected but not found. For instance, a C-type constant coded with no ending ' is flagged this way.

AS123 MISSING OPERAND
The instruction requires an operand, but it is not specified.

AS124 LABEL REQUIRED
An instruction requiring a label, such as a DSECT, is coded without one.

AS126 RELOCATABLE EXPRESSION REQUIRED
An absolute expression or term is used where a relocatable one is required by ASSIST, such as in the first operand of a USING. Also, this message may appear if the final relocatability attribute of the value in an address constant is that of a symbol in a DSECT.

AS128 ILLEGAL START CARD
The START card flagged is coded with one or more statements other than listing controls or comments appearing before it.

AS129 ILLEGAL USE OF LITERAL
The literal constant appears in the receiving field of an instruction which modifies storage.

AS130 UNDEFINED SYMBOL
The symbol shown is either completely undefined, or has not been already defined when it is required to be. Symbols used in ORG instructions or in constant lengths or duplication factors must be defined before they are used.

AS131 UNRESOLVED EXTERNAL REFERENCE
The symbol used in a V-type constant is not defined in the assembly, or is defined but not declared a CSECT or ENTRY. ASSIST does not link multiple assemblies, so this is an error.

AS132 ILLEGAL CHARACTER
The character flagged is either not in the set of acceptable characters, or is used in an illegal way.

AS135 SYNTAX
The character flagged is improperly used. This catchall message is given by the general expression evaluator when it does not find what is expected during a scan.

Appendix D: Runtime Exceptions

This appendix contains basic descriptions of the types of fatal exceptions that cause program execution to terminate. The following exceptions are supported: Addressing exception; data exception; decimal-overflow exception; decimal-divide exception; fixed-point overflow exception; fixed-point divide exception; operation exception; protection exception; and, specification exception. Table 4 summarizes which types of errors are possible for each instruction (see Appendix B).

An addressing exception (A) occurs when attempting to reference an address outside of actual memory. A data exception (D) is the result of an attempt to manipulate a field that should contain a valid packed-decimal number but does not. A decimal-overflow exception (DO) occurs when the result of a decimal arithmetic instruction cannot be represented in the receiving field. A decimal-divide exception (DD) occurs when the quotient from the execution of a DP statement too large to be represented in the allotted space or when an attempt is made to divide by zero. A fixed-point overflow exception (FO) occurs when the result of an arithmetic operation cannot be represented in a 32-bit word. A fixed-point divide exception (FD) occurs when the quotient of D or DR cannot be represented in a 32-bit word. This error also occurs in the event of division by zero. An operation exception (O) occurs when an attempt is made to execute a memory value that is not a valid instruction. An operation exception can occur if a constant is included in the middle of a routine, if a section of a program is altered during execution, or if a branch address is incorrect. A protection (P) exception occurs when an executed instruction attempts to access or modify memory outside the limits of the program. A specification exception (S) occurs when boundary alignment is unobserved or when an unexpected value is encountered in an index register.

Table 4: Fatal exceptions for each instruction.

Mnemonic	A	D	DO	DD	FD	FO	P	S
A	X				X		X	X
AP	X	X	X				X	
AR					X			
BAL								
BALR								
BC								
BCR								
BCT								
BCTR								
BXH								
BXLE								
C	X						X	X
CLC	X						X	
CLI	X						X	
CP	X	X					X	
CR								
D	X					X	X	X
DP	X	X		X			X	X

DR						X		X
ED	X	X					X	
EDMK	X	X					X	
L	X						X	X
LA								
LM	X						X	X
LR								
M	X						X	X
MP	X	X					X	X
MR								X
MVC	X						X	
MVI	X						X	
N	X						X	X
NR								
O	X						X	X
OR								
PACK	X						X	
S	X				X		X	X
SP	X	X	X				X	
SR					X			
ST	X						X	X
STM	X						X	X
UNPK	X						X	
ZAP	X	X	X				X	