General Notice

When using this document, keep the following in mind:

1. This document is confidential. By accepting this document you acknowledge that you are bound by the terms set forth in the non-disclosure and confidentiality agreement signed separately and /in the possession of SEGA. If you have not signed such a non-disclosure agreement, please contact SEGA immediately and return this document to SEGA.

2. This document may include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new versions of the document. SEGA may make improvements and/or changes in the product(s) and/or the program(s) described in this document at any time.

3. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without SEGA'S written permission. Request for copies of this document and for technical information about SEGA products must be made to your authorized SEGA Technical Services representative.

4. No license is granted by implication or otherwise under any patents, copyrights, trademarks, or other intellectual property rights of SEGA Enterprises, Ltd., SEGA of America, Inc., or any third party.

5. Software, circuitry, and other examples described herein are meant merely to indicate the characteristics and performance of SEGA's products. SEGA assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples describe herein.

6. It is possible that this document may contain reference to, or information about, SEGA products (development hardware/software) or services that are not provided in countries other than Japan. Such references/information must not be construed to mean that SEGA intends to provide such SEGA products or services in countries other than Japan. Any reference of a SEGA licensed product/program in this document is not intended to state or simply that you can use only SEGA's licensed products/programs. Any functionally equivalent hardware/software can be used instead.

7. SEGA will not be held responsible for any damage to the user that may result from accidents or any other reasons during operation of the user's equipment, or programs according to this document.

NOTE: A reader's comment/correction form is provided with this document. Please address comments to :

SEGA of America, Inc., Developer Technical Support (att. Evelyn Merritt)
150 Shoreline Drive, Redwood City, CA 94065

SEGA may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

SEGA OF AMERICA, INC.
Consumer Products Division

Microcomputer Developing Integrated Environment for Macintosh

Functions Specifications
Vol. 1

ST-80-R2-050994

Contents

READER CORRECTION/COMMENT SHEET

Keep us updated!

If you should come across any incorrect or outdated information while reading through the attached document, or come up with any questions or comments, please let us know so that we can make the required changes in subsequent revisions. Simply fill out all information below and return this form to the Developer Technical Support Manager at the address below. Please make more copies of this form if more space is needed. Thank you.

General Information:

Your Name _____ **Phone** _____

Document number ST-82-R2-050994 **Date** _____

Document name Microcomputer Developing Integrated Environment for Macintosh

Corrections:

Chpt.	pg. #	Correction

Questions/comments: _____

Where to send your corrections:

Fax: (415) 802-3963
Attn: Manager,
Developer Technical Support

Mail: SEGA OF AMERICA
Attn: Manager,
Developer Technical Support
275 Shoreline Dr. Ste 500
Redwood City, CA 94065

1.0 Preface

This product, equipped with a text editor, assembler, and object linker that operate on Apple computers, is an integrated environment software that supports software development of microcomputers. Using the Macintosh's superior interface, the software of two types of standard CPU, that is the Z80 and the MC68000, can consistently be developed beginning with the creation of the source code and ending with the production of the object module.

2.0 About the Operation Environment

- Model — Macintosh series that can operate Kanji Talk 7.1, with a CPU that is 68030 or later
- RAM capacity — minimum of 8 Mbytes required
- HD capacity — minimum of 300 bytes required

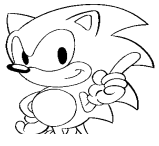
Installation

Do the following to install the software:

1. Insert the SDSS VER 1.0 program disk in the floppy disk drive of the Macintosh.
2. Copy all files from the SDSS VER 1.0 program disk on to the hard disk. (You do not need to create a special folder.)
3. Eject the SDSS VER 1.0 program disk.

Starting Up SDSS

An icon like the one below should appear on the hard disk. Double click this icon to start up the program.



3.0 Operation Overview of Integrated Environment

SDSS Integrated Environment Software can do the following:

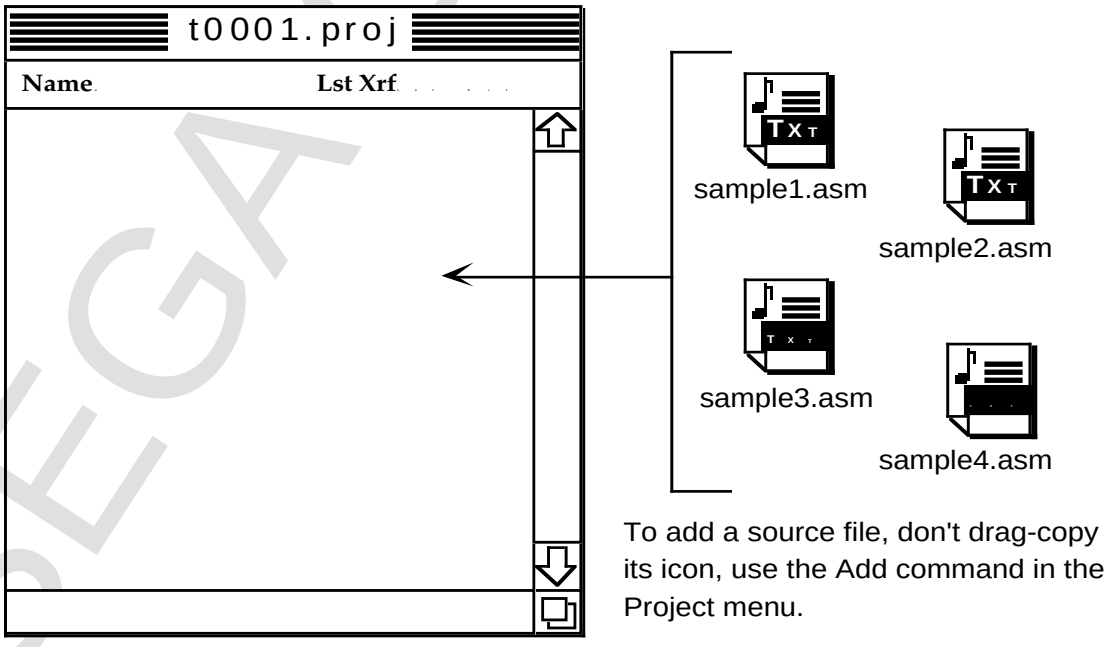
- Create and edit (text edit) source files
- Manage source file projects (text batch management)
- Assemble source files (Z80, MC68000)
- Link object files (creates load module)

Creating and Editing Source Files

Source (text) files can be created in the SDSS program. The file created appears as an icon that looks like the one at the left. This icon tells you that the file was created by the SDSS program. Because these files have text attributes they can be edited by other text editors. Further, the SDSS program can be started up by double clicking on this icon. The text window of the file will open by double clicking the icon.

Managing a Source File Project

In order to perform the assemble/link process, a source file must be added to the project.



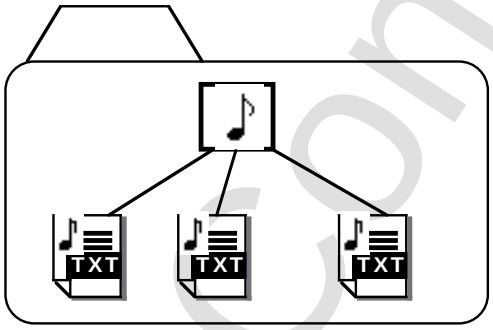
When adding a source file, the source name being displayed, as shown in the example to the right, lets you know that the file has been added. In other words, only information is stored in the project (not the source file substance.)

The project is treated as one file, and an icon like the one to the left appears.

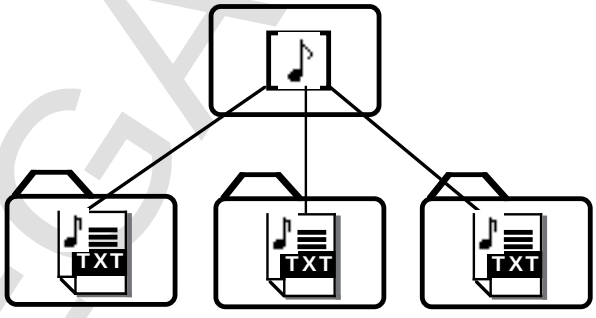
t0001.proj

Name	Lst	Xrf
sample1.asm	ON	ON
sample2.asm	OFF	OFF
sample3.asm	OFF	OFF
sample4.asm	ON	OFF

As shown in the examples below, the relationship between the project and source has two possible configurations. In example 1, moving and managing files becomes simple.

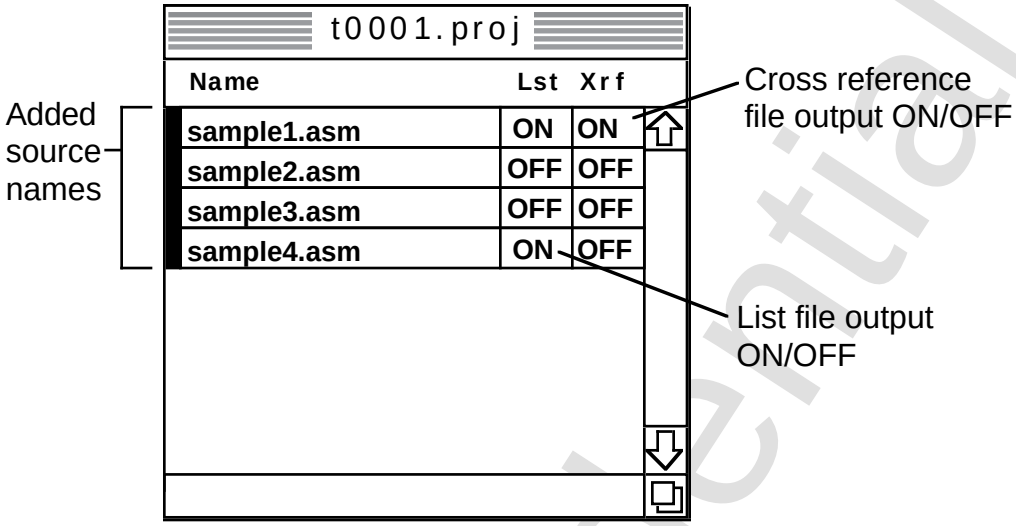


Example 1



Example 2

Note: Care must be taken in moving to other machines.
Positions of source files stored in projects are not guaranteed in the movement time of other models. Here the source is added again to the project.

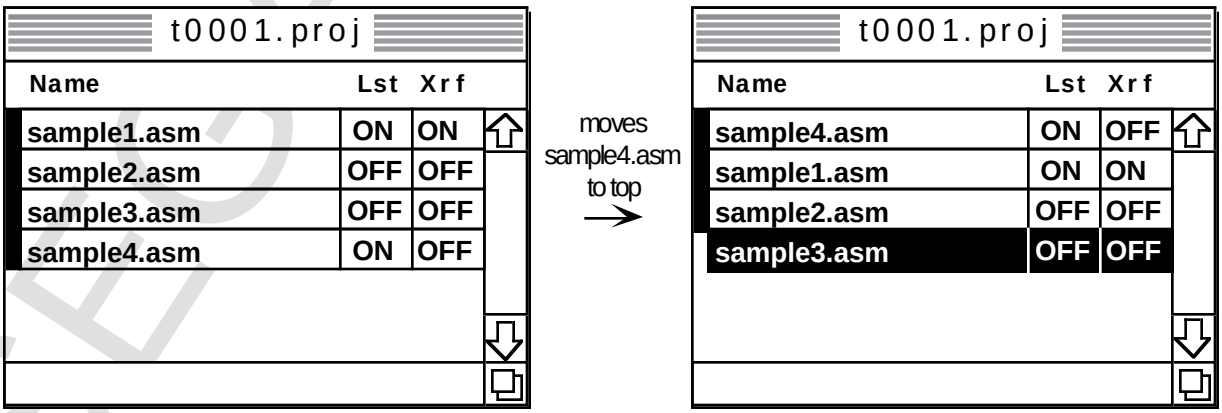


The cross-reference and list file output switches when adding a project is:

List File (Lst)	ON
Cross Reference (Xrf)	OFF

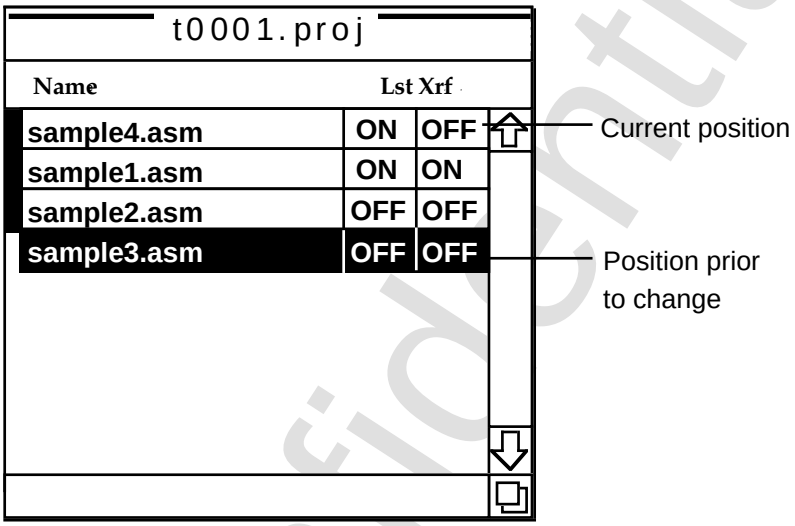
When this is to be changed, click the letters of the switch to be changed (ON or OFF) with the mouse.

You can also change the order of the sources. Because assembling after a file that produced an assembler error cannot be done, it is convenient to check single files. Use the mouse to drag a file whose order you want to change, then release the mouse button after you have moved the file to the desired location.

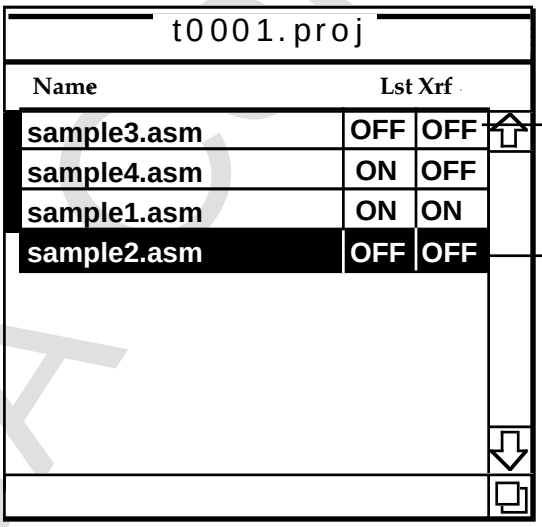


Do the following by changing the sequence of the source while adding.

Clicking the scroll bar rotates between the current position and the original position of the file moved



Direction of rotation is fixed



Source File Assemble

Assemble is executed by the Assemble or Assemble All Files command in the Project Menu. Both commands assemble by order sources added to the project.

- Assemble — Executed only within a source that has been added to a project with no object file or with a source file that has been renewed.
- Assemble All Files — Assembles all sources added to the project file. (Added in order)

If an assemble error occurs for any command, files cannot be assembled after assemble execution is stopped by the file. "Assemble Complete !!" is displayed when a file is succeessfully assembled; but if an error occurs, that information and the error line are displayed in the information window.

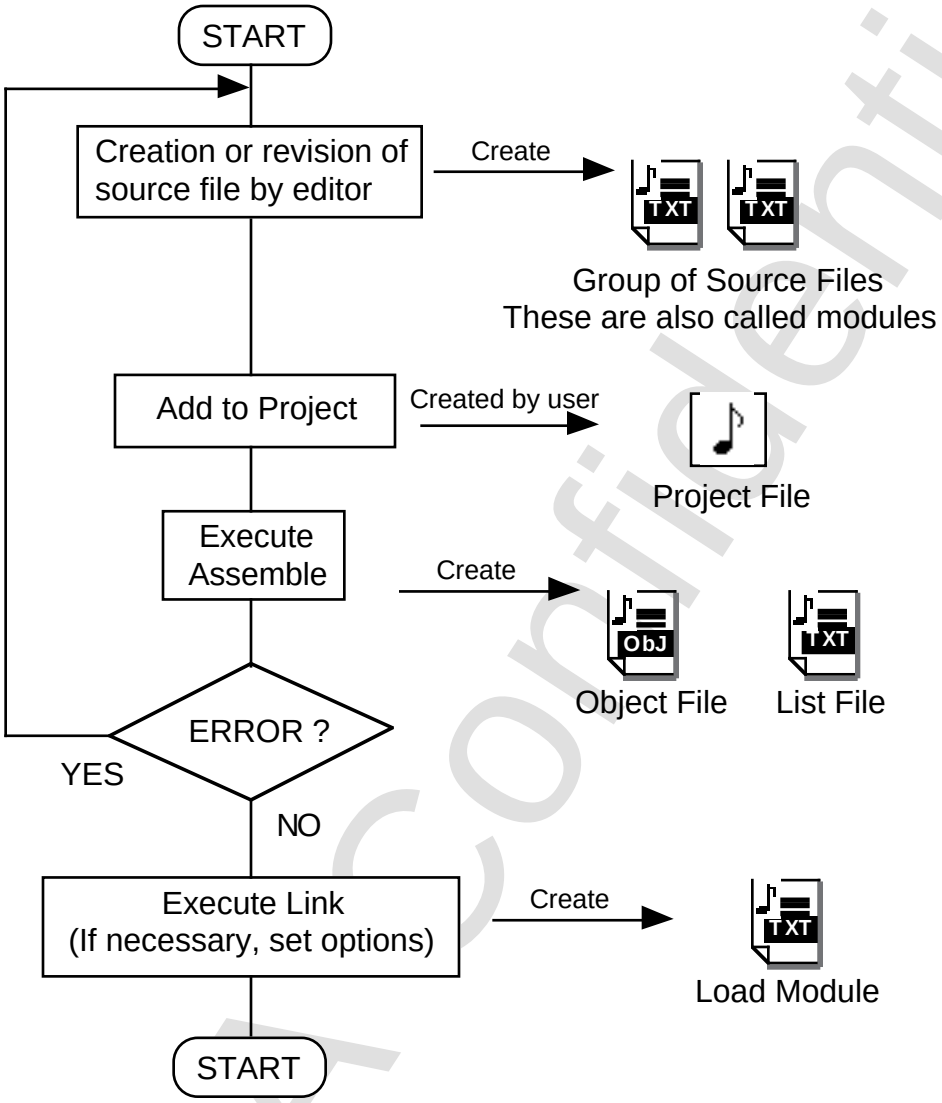
Object File Link

Assemble is executed by the Link command in the Project menu. This creates a load module from the obtained object file set by assembling the source file added to the project and can set the output format (Motorola S, Intel HEX, three types of binary), the basic address, etc., by command.

Note: The output formats of the load module has been added to the end of this manual.

4.0 Flowchart of a Unified Environmental Software (SDSS)

This flowchart simply illustrates what has been written up to now in this manual.

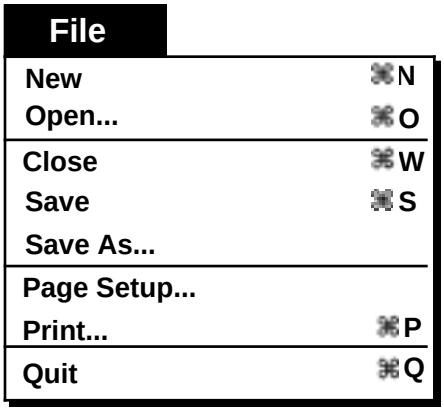


- Source File (Module) — Files describing programs and data are called modules when arranged in each function.
- Object (Module) — Files that are created and assembled from source files.
- Load Module — Created through linking, these files are a combination of object files created from source files being added to a project. This file is in a format that can be executed by debug or target.

5.0 MENU REFERENCE

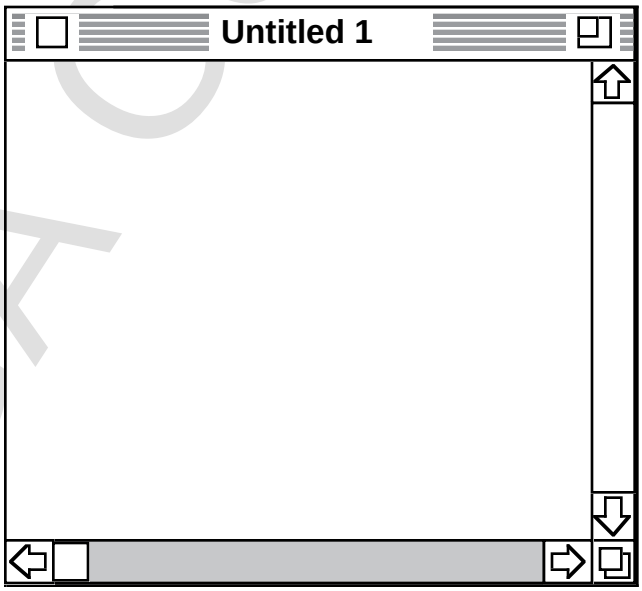
File Menu

The source (text) create, save, and print commands, and commands to open and close pre-existing text are in the File menu.

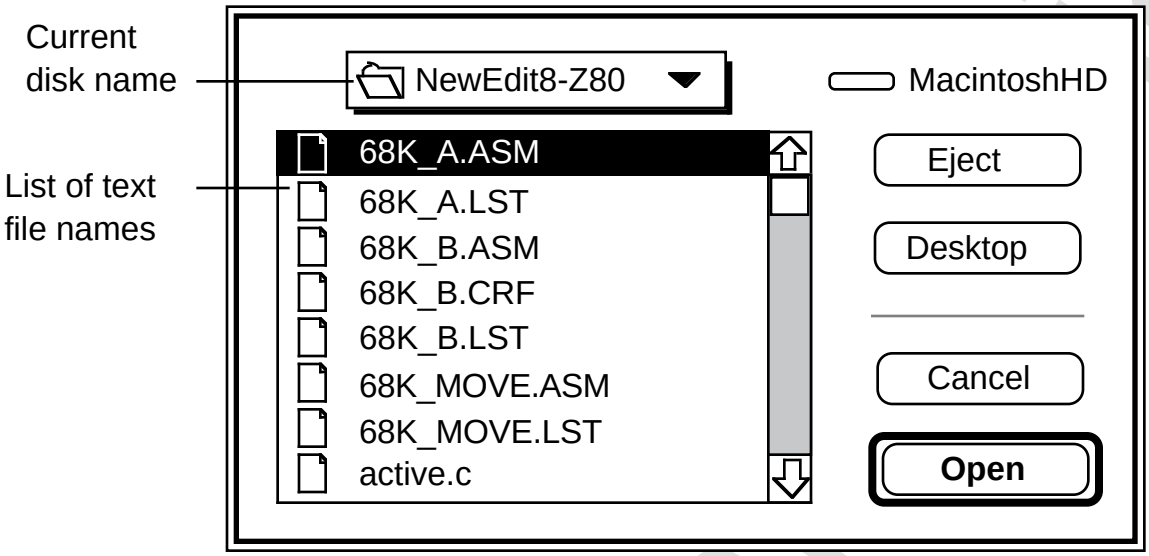


New (Shortcut: Command-N)

Creates and opens a new text window. The name of the window at this time will be "untitled".

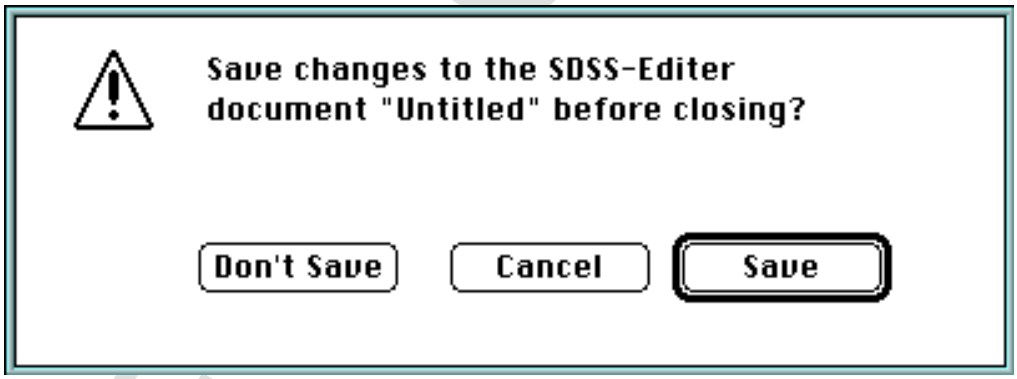


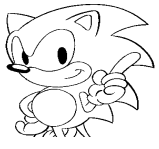
Open (Shortcut: Command-O)
Opens pre-existing text. The Macintosh standard file dialog box is displayed and you can select the file you want to open from inside the box.



The file names displayed here are limited to TEXT file types.

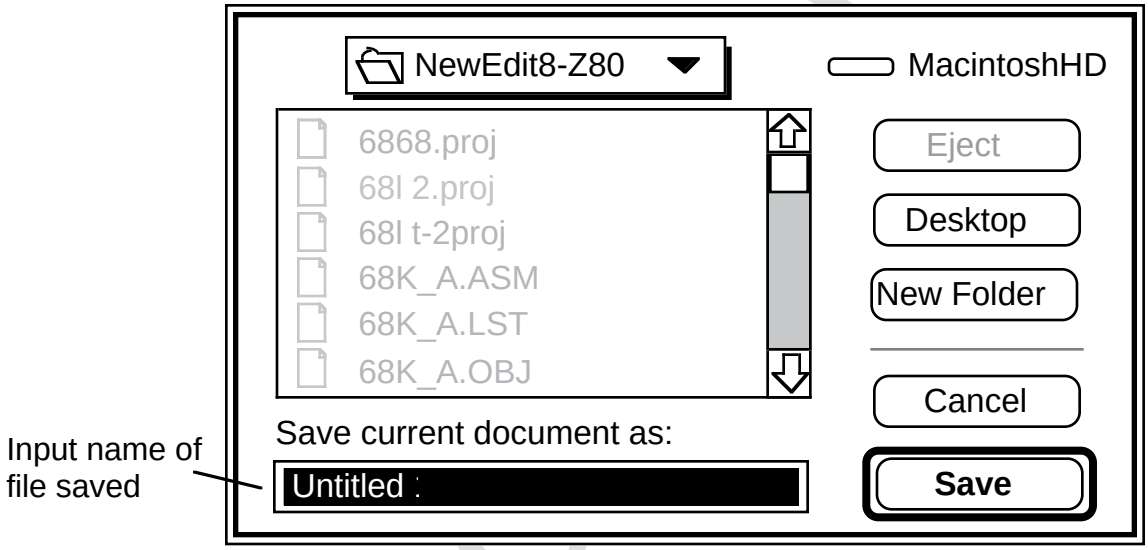
Close (Shortcut: Command-W)
Closes the text window currently being edited. If changes have been made in the text, the following save dialog box will appear.





Save (Shortcut: Command-S)
Writes to the file the text currently being edited. The text edit keeps the window as it is, without interrupting.

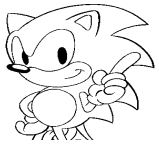
Save As ...
Save As lets you save currently edited text into the file while changing the file name to be different from the current window title. The file name is specified within the Macintosh standard file save dialog box, which appears as below. The text edit keeps the window as it is, without interrupting.



Page Setup ...
Sets the page size and options for printing.
The dialog box displayed here will be different depending on the type of printer you selected in Chooser under the Apple menu. Please refer to the printer manual.

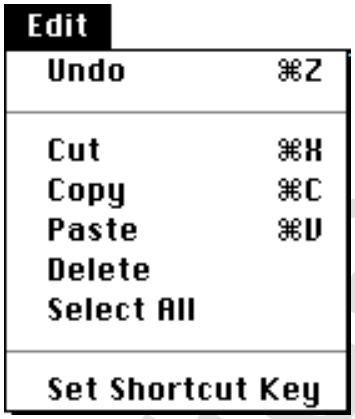
Print (Shortcut: Command-P)
Prints the contents of the text window that is currently open.

Quit (Shortcut: Command-Q)
Quits SDSS. Automatically closes the text and project windows currently open. (If any changes have been made, the confirm dialog box is displayed at the same time as "close.")



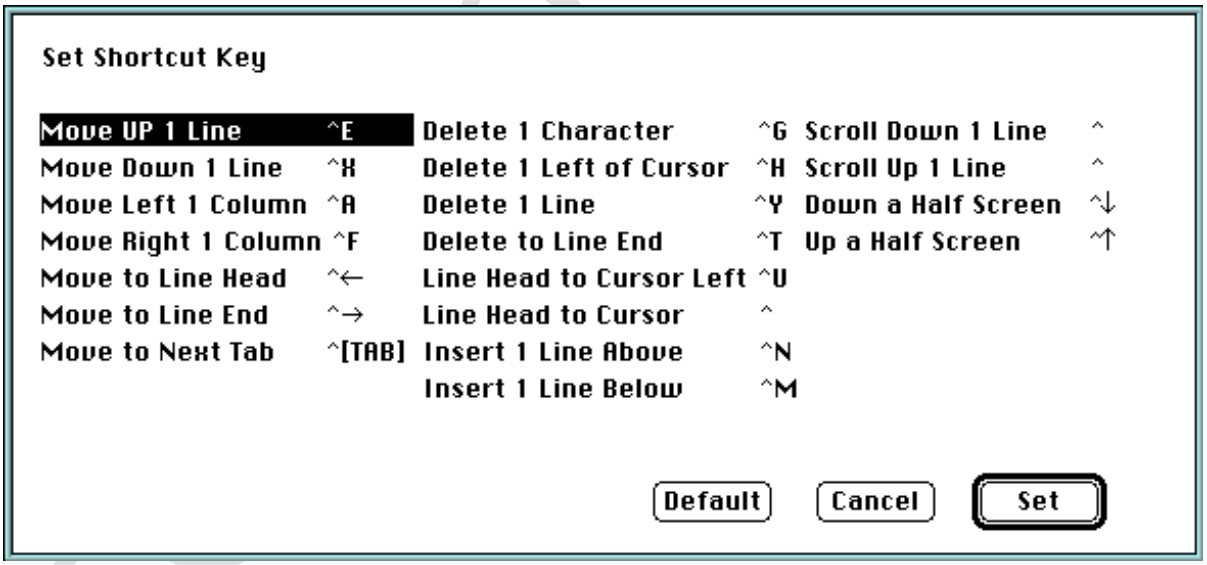
Edit Menu

The Edit menu has standard Macintosh edit commands including cut and paste, and customized functions keys used by the editor. Since Undo, Cut, Copy, Paste, and Select All are all standard Macintosh commands, they will not be explained here.



Set Shortcut Key

A shortcut key is assigned to each command of the Special menu, and can be changed using this command.



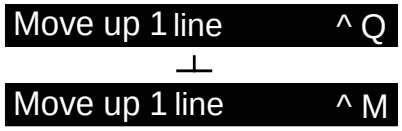
Default:	Returns to default value
Cancel:	Cancels changes
Set:	Quits setwindow

How to Set Shortcut Keys

The following describes how to set a shortcut key.

(1) Click the mouse button on the item you want to set.

(2) The selected item will become highlighted. Pressing keys changes the shortcut key setting display (keys are limited to letters, arrows, **TAB**, and **DEL** keys).



When M is pressed

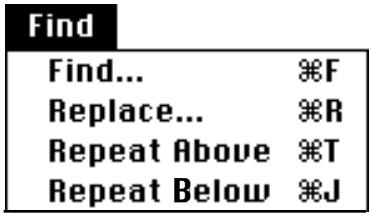
(3) Changes take effect when you click **Set**.
The **Cancel** button cancels any changes (returns to the conditions prior to the change).
The Default button sets values to their default value.

Notes:

- The ^ symbol refers to the **Control** key. (You cannot change to other keys such as the **Option** key).
- When more than one item is assigned to the same key, only the first setting is valid. Other items set can not be used.

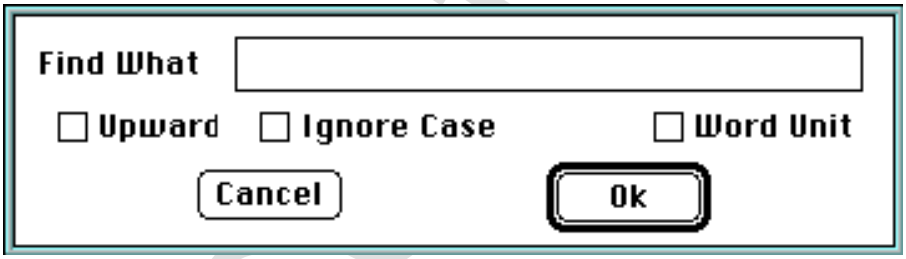
Find Menu

The Find menu has a character string find command and character string replace command that operates in a text window.



Find (Shortcut: Command-F)

This function locates any character string from within the active text window. By inputting the required item, the dialog box below appears and searches for character strings that meet the conditions. If that item exists, the position of the screen is adjusted so that the line with the character string is displayed and the character string that is found is highlighted.



Find what —— Area for inputting the character string you want to find.

Find options —— There are three options, clicking each small square button activates each function. The options are:

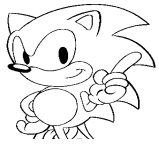
- Upward: Searches upward from the current line. (Normal search is downward)
- Ignore case: Searches for same character alignment regardless of upper or lower case.

• Word Unit: Handles a character string as a single unit.

In the following example, when WORD is input and searched, only WORD in the first line is found and displayed.

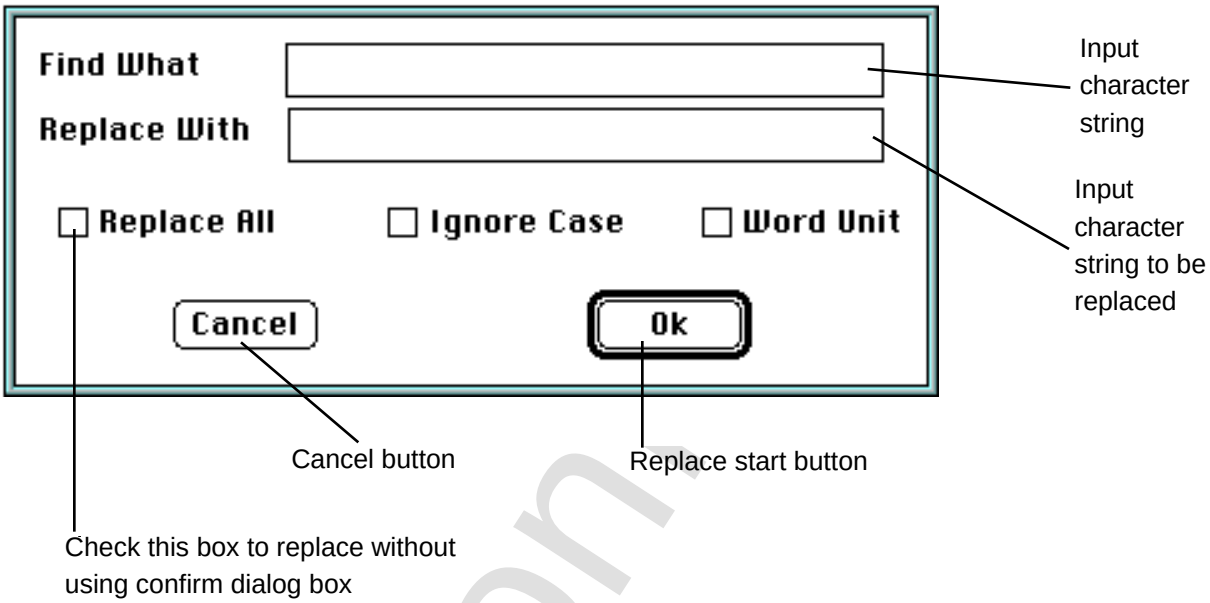
Example:

```
WORD    A
WORD1   B
WORD2   C
WORD3   D
```

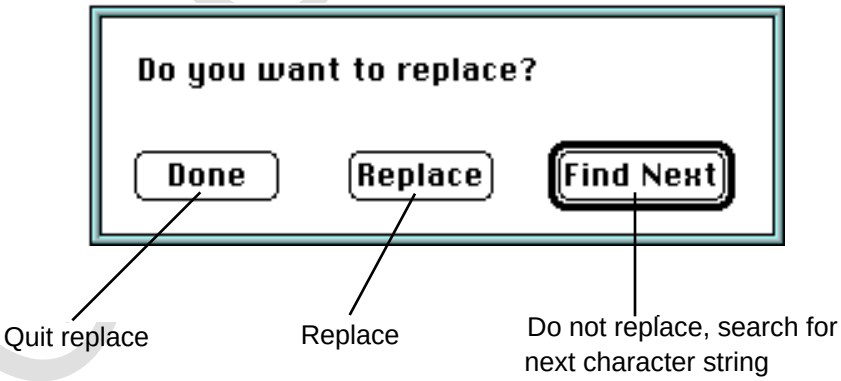



Replace (Shortcut: Command-R)

This is a function that searches any character string from among active text windows and replaces it with a specified character string. By setting the necessary items in the dialog box, the character strings you want to replace is searched and replaced if found. (If the "Replace all" box is not checked, the confirm dialog box will appear for each character string.)



When **Find next** is selected, the replace confirmation dialog box appears and confirms replacement.



Repeat Above (Shortcut: Command-T)
This is the Find repeat command that searches text upward from the current line (towards the beginning of the file). Character strings to be searched are set by the Find command. In short, this command lets you search continuously without having to input the character string to be searched over and over again.

Repeat Below (Shortcut: Command-J)
This is the Find repeat command that searches text down from the current line (towards the end of the file). Character strings to be searched are set by the Find command. In short, this command lets you search continuously without having to input the character string to be searched over and over again.

Jump Menu

The Jump menu has six commands for quickly moving the cursor in the active text window to a desired location.

Top Line

Moves cursor to the first line of a file.

Set Mark

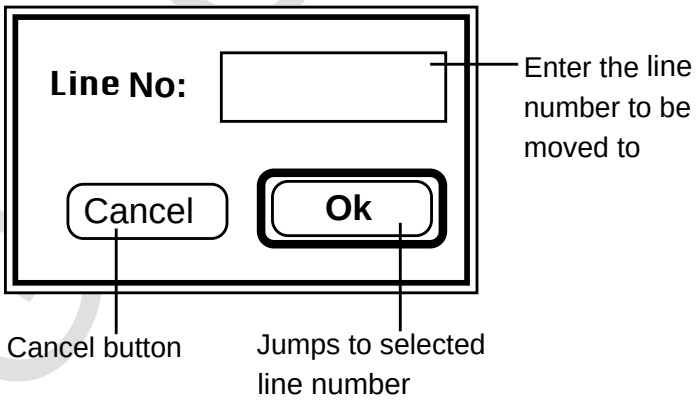
Puts a mark at the current insertion point (cursor). Although nothing is displayed, the Set Mark command can move the cursor to any line specified by the mark from any line.

Marked Line

The Marked Line command can move the cursor to lines specified by the mark.

Specify Line...

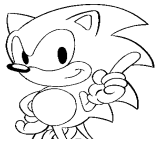
You can move the cursor to the "Line No:" as in the dialog box below.



Previous Line
Cursor jumps to the line before the current position.

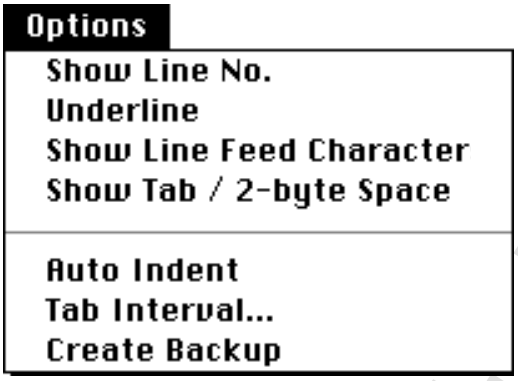
For example, you can move to the first line from the bottom line by using this command.

Bottom Line
Moves the cursor to the bottom line in a file.



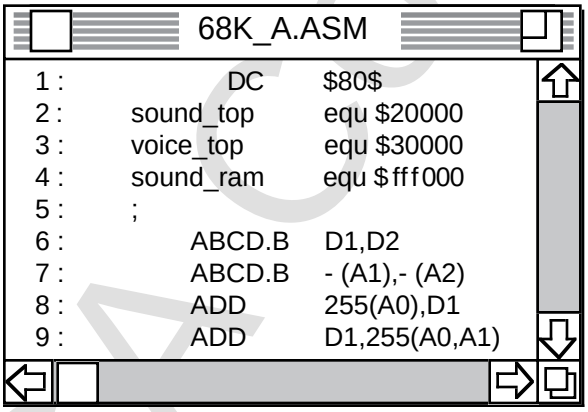
Options Menu

The Options menu has seven commands that define the operating environment of the active text window. These commands function independently in each window. A check mark is displayed at each selected function for easy reference.



Show Line No.

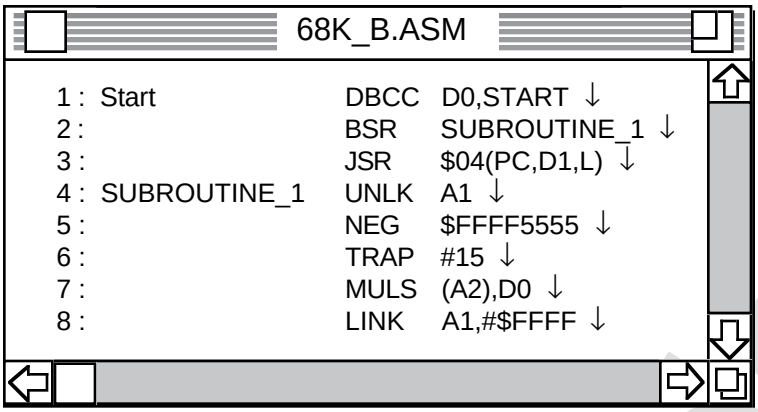
Displays the line number in the lead of each text line.



Underline

Underlines cursor line.

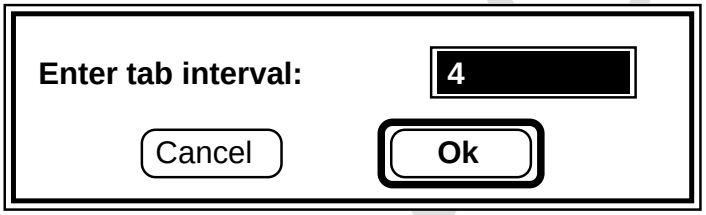
Show Line Feed Character
Displays an arrow (↓) at the spot of the line feed character.



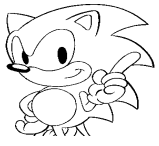
Show Tab/2-Byte Space
Displays spaces at locations where a tab code space exists.

Auto Indent
When indenting a line, indentation of the next line follows the indentation of the line before it.

Tab Interval
Sets how many characters will be jumped when TAB is set. The default value is four. (Half size spaces of four characters)



Create Backup
Creates a backup file when a file is opened.



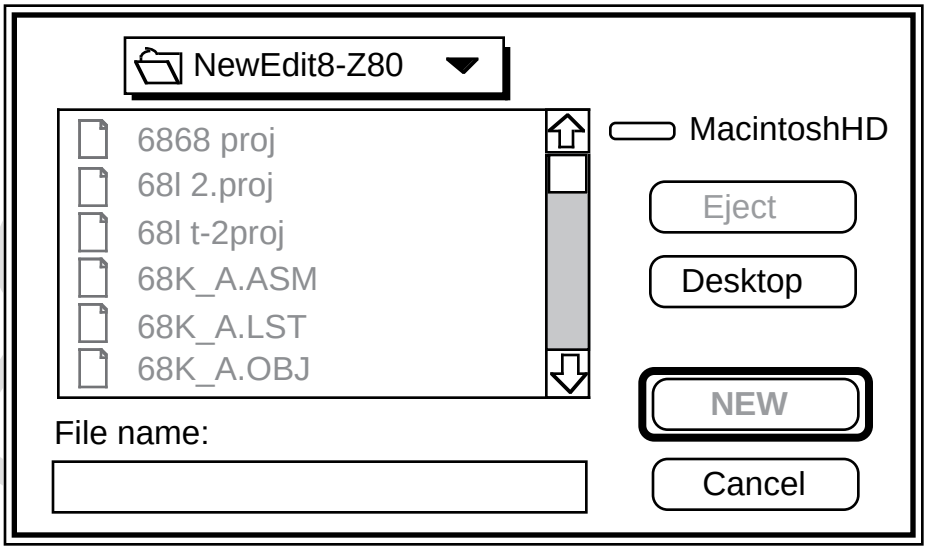
Project Menu

Commands within the Project menu include managing the project window and executing the assemble and linking commands. This menu can do most operations except text editing.

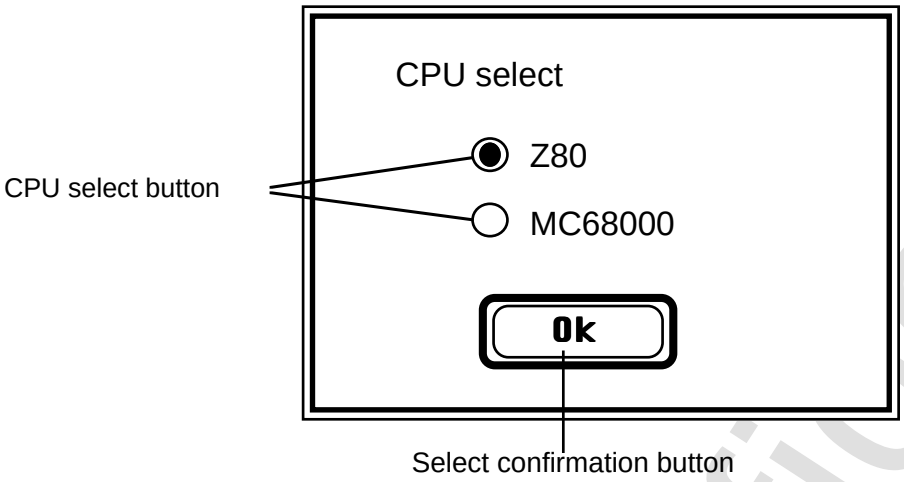
New Project

Opens a new project window.

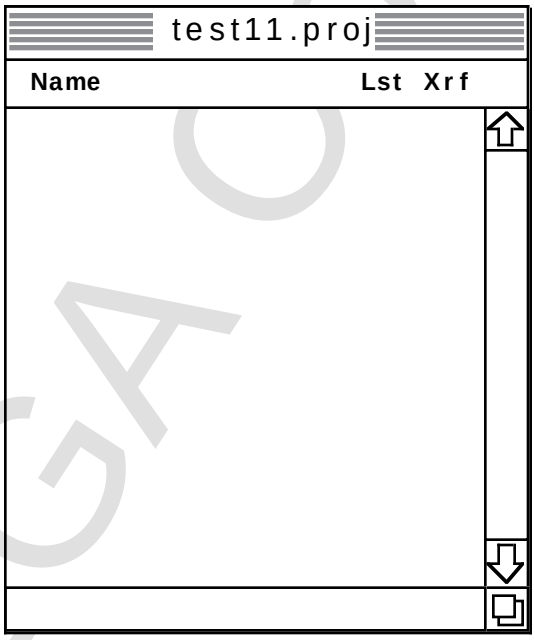
This command cannot be used when a project is open. In the example below, type the project name to be displayed in the dialog box. The extension ".PROJ" is automatically added to the file name.



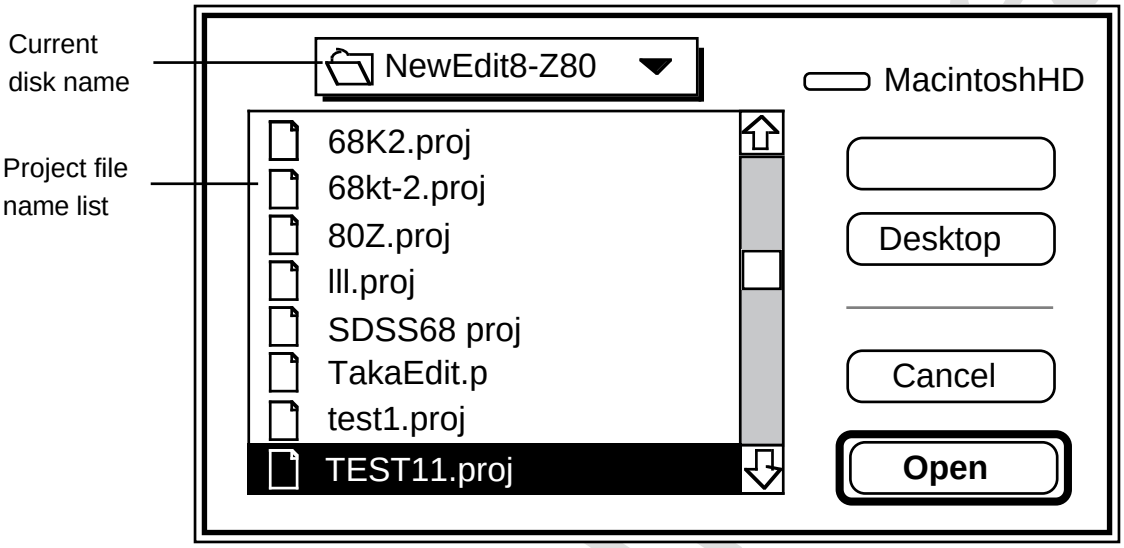
When the dialog box that asks what type of CPU you want to use for the open project appears, click the CPU you want to use. One project corresponds to only one CPU.



The project window is opened when the selection is confirmed.



Read
Reads projects that already exist. When the file select dialog box appears, you can designate which file to open.



The file names displayed above are limited to "PROJ" files.

Save
Saves projects currently open. You can continue without closing the project window.

Front
This command activates the project window. This command can be used when the text window is hidden behind other windows.

Assemble (Shortcut: Command-K)
This command assembles sources added to the project. However, it will not be assembled in the following cases.

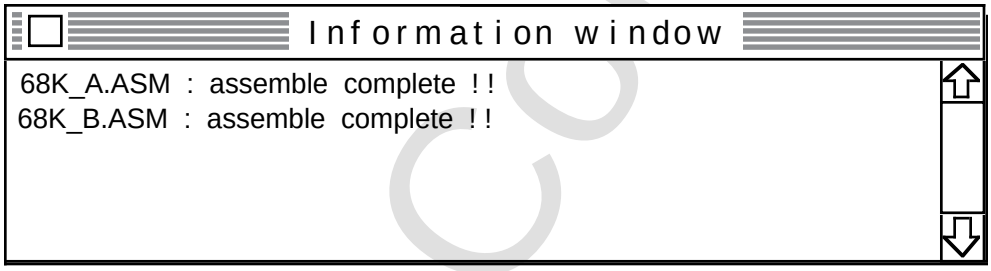
- When the object has already been created and there is no change in the source.
- When there is an error in the file assembled previously.

Sources will not be assembled after an error occurs. The source object file that produced the error is deleted at this time.

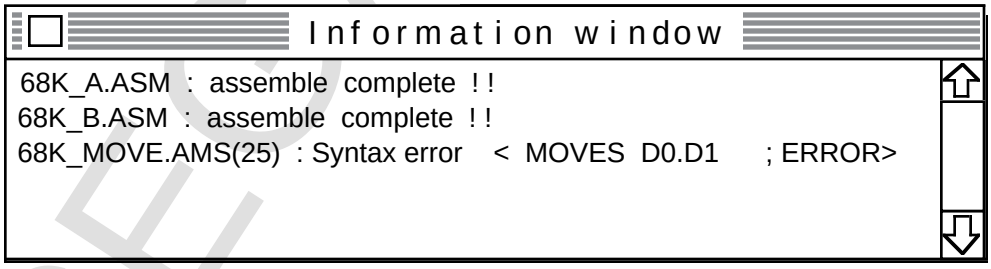
AssembleAll Files
This command assembles sources added to the project. All sources are assembled without being checked for an Assemble command source change. Sources will not be assembled after an error occurs.

DisplayingAssembled Results
An information window opens when assemble is executed and performs the following information.

- After normal completion, "assemble complete !!" is displayed in each file.



When an error occurs, the line where the error occurred and the error type are displayed.

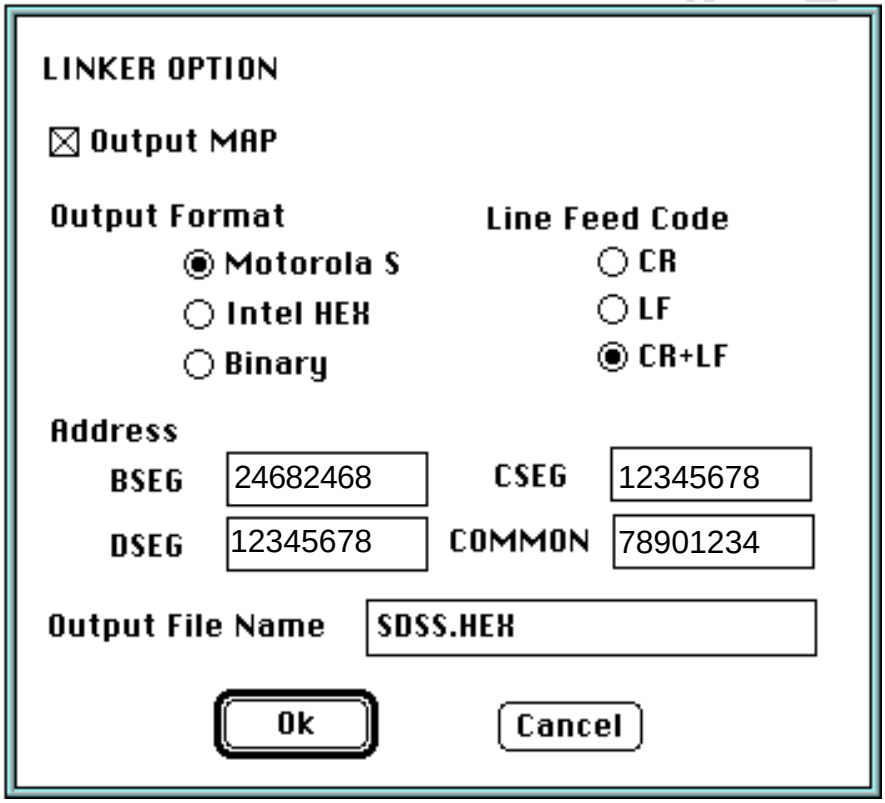


Link (Shortcut; Command-L)

This command links source objects added to the project; it also creates a load module file (file name is set by the Option command). The results of the link are reported in the information window the same as when executing assemble.

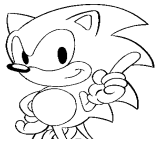
Option (Shortcut; Command-Y)

This command sets all parameters used during the link process. The dialog box shown below is used for making the settings.



- Output MAP — Checks for when a MAP file is required. At the end of a normal link, the file with MAP as an extension to the output file name is output.
- Output Format — Selects the format of the HEX file to be output. Motorola S, Intel HEX, and Binary are exclusive of each other and you can selected only one.

- Line Feed Code — Selects the line feed code of the HEX file to be output. You can select one of the following: CR(0x0d), LF(0x0a), or CR+LF(0x0d0a)
- Address — The lead address of each segment is set as hexadecimal numbers. Eight digit input is possible, but Z80 is valid for only the lower four digits.
- Output File Name — Sets the HEX file name.



Source Menu

The Source menu adds sources to a project, and contains a delete command and binary file edit command.

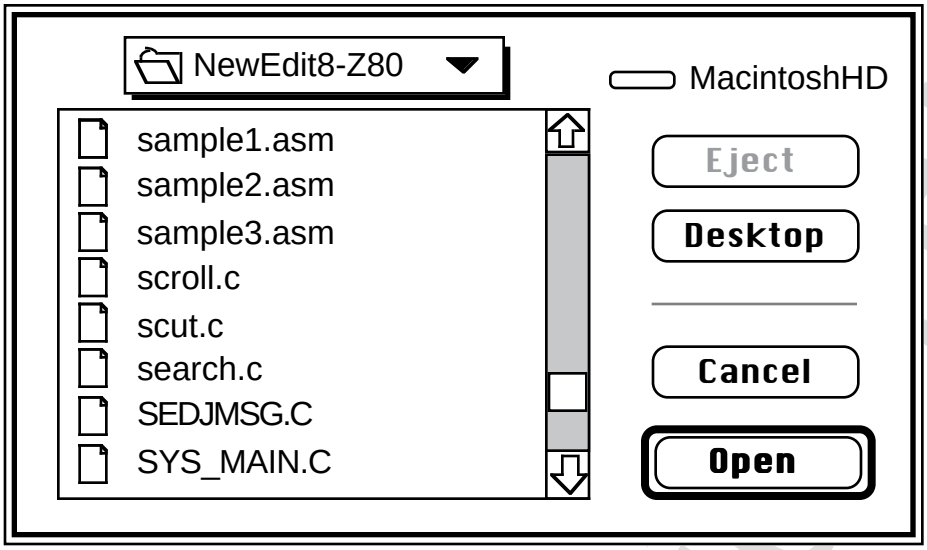
Add

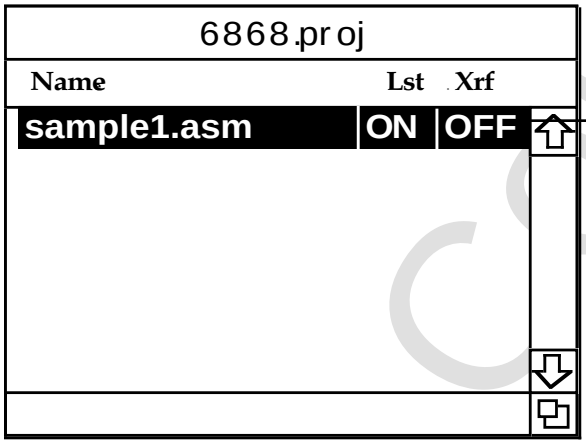
The Add command adds sources to a project. Text is added when Add is used from an active text window. Also, if you use the project window when it is active, the file select dialog box will appear and the file you selected will be added.

Note: The following files cannot be added to a project,

- Files that have already been added
- New files (text without a title).
  (An alert box like the one below will appear.)

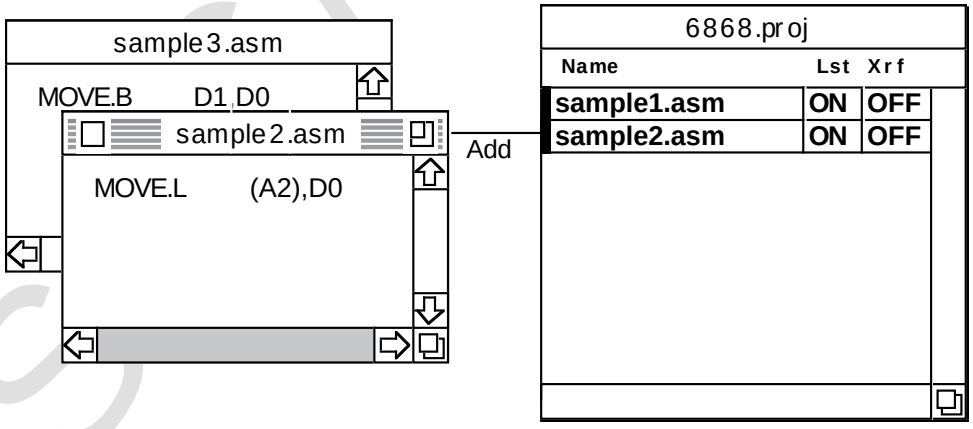
- Use Add when the project window is active.





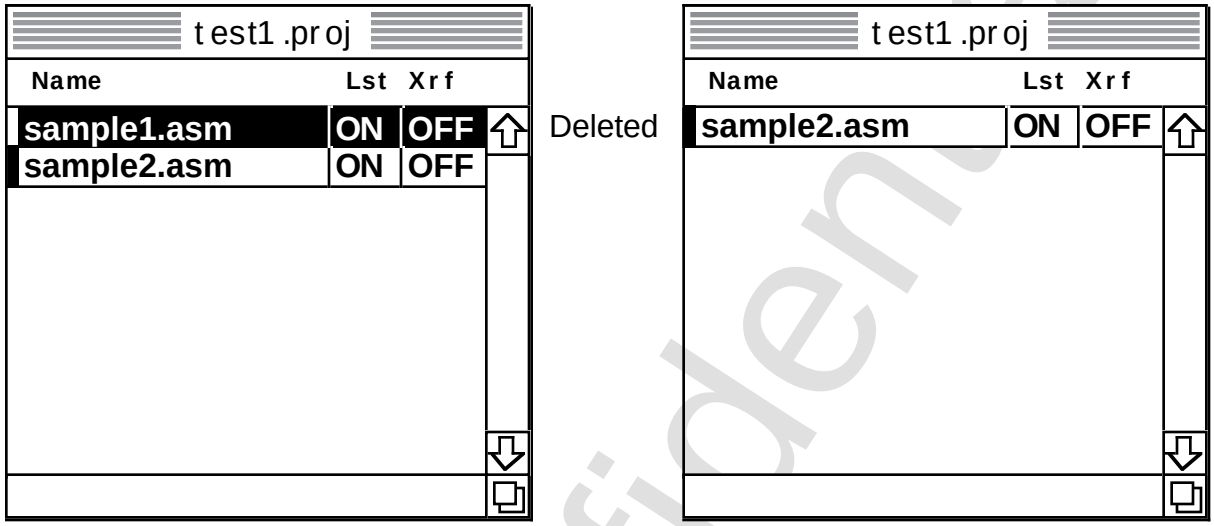
Files selected by the file select dialog box are added to the project window.

- Use Add when the project window is active.



Delete

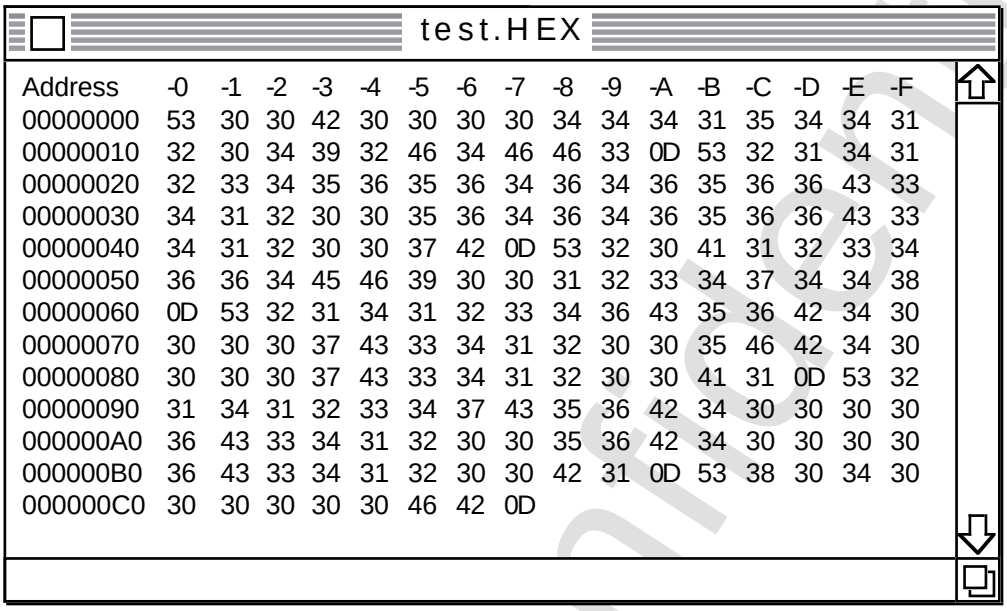
You can remove a source from a project by specifiying and executing the source added to the project.



Deleting a file is done by clicking on the file name in the project window. The file selected will be highlighted.

Edit Binary File

This is an expansion command that will be available in the future, but with today's version. In its place, however, is the application **binedit** . This application is for display and edit of files by a hexadecimal binary code. Operations like the one shown below are possible.



Address	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
00000000	53	30	30	42	30	30	30	30	34	34	34	31	35	34	34	31
00000010	32	30	34	39	32	46	34	46	46	33	0D	53	32	31	34	31
00000020	32	33	34	35	36	35	36	34	36	34	36	35	36	36	43	33
00000030	34	31	32	30	30	35	36	34	36	34	36	35	36	36	43	33
00000040	34	31	32	30	30	37	42	0D	53	32	30	41	31	32	33	34
00000050	36	36	34	45	46	39	30	30	31	32	33	34	37	34	34	38
00000060	0D	53	32	31	34	31	32	33	34	36	43	35	36	42	34	30
00000070	30	30	30	37	43	33	34	31	32	30	30	35	46	42	34	30
00000080	30	30	30	37	43	33	34	31	32	30	30	41	31	0D	53	32
00000090	31	34	31	32	33	34	37	43	35	36	42	34	30	30	30	30
000000A0	36	43	33	34	31	32	30	30	35	36	42	34	30	30	30	30
000000B0	36	43	33	34	31	32	30	30	42	31	0D	53	38	30	34	30
000000C0	30	30	30	30	30	46	42	0D								

Keys that can be used:

0 ~ 9 Inputs the number of the current position of the cursor. Inputting overwrites older data.

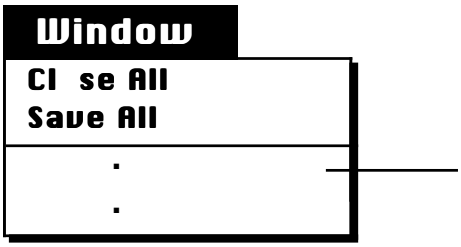
A ~ F A hexadecimal number can be input in the current position of the cursor. Inputting overwrites older data.

DELETE Deletes one byte at the current position of the cursor.

I Inputs 00 in single bytes in the current position of the cursor

Window Menu

The Window menu has a command that closes all text windows curently open. It also has a command for changing text windows.



Close All

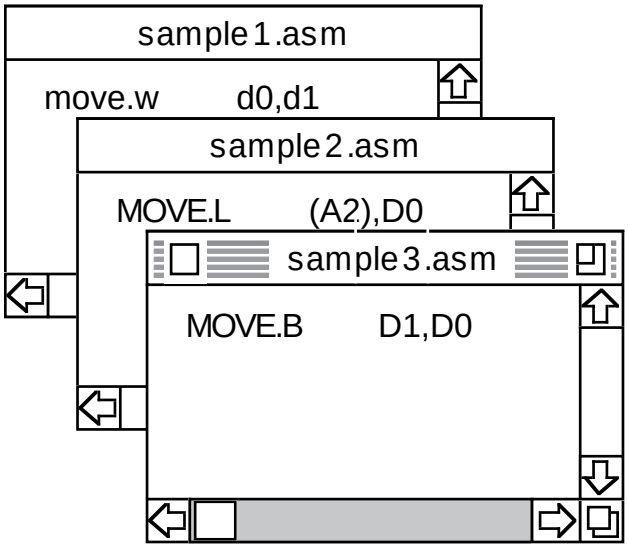
Closes all text windows that are currently open. Windows are closed automatically by selecting this command. (However, if there are changes, a dialog box will appear allowing you to save the changes before closing.)

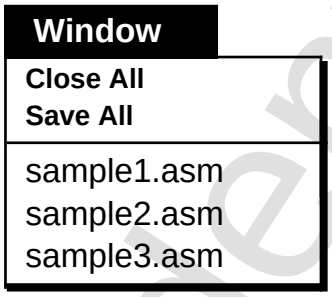
Save All

Saves then closes all text windows that are currently open. Windows are closed automatically by selecting this command.

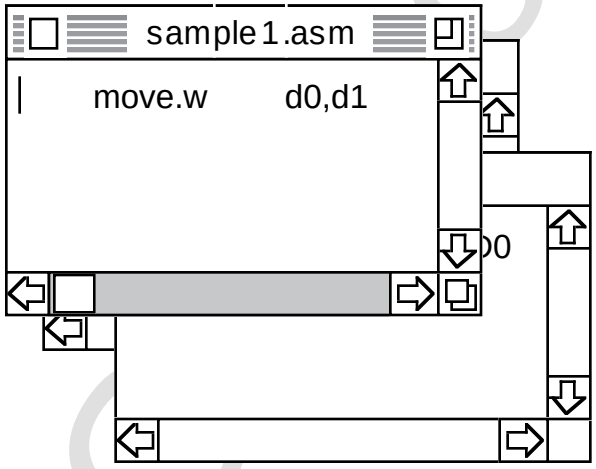
Change Text Window

The titles of all text windows currently open are shown in the menu. The windows selected from the menu become active.



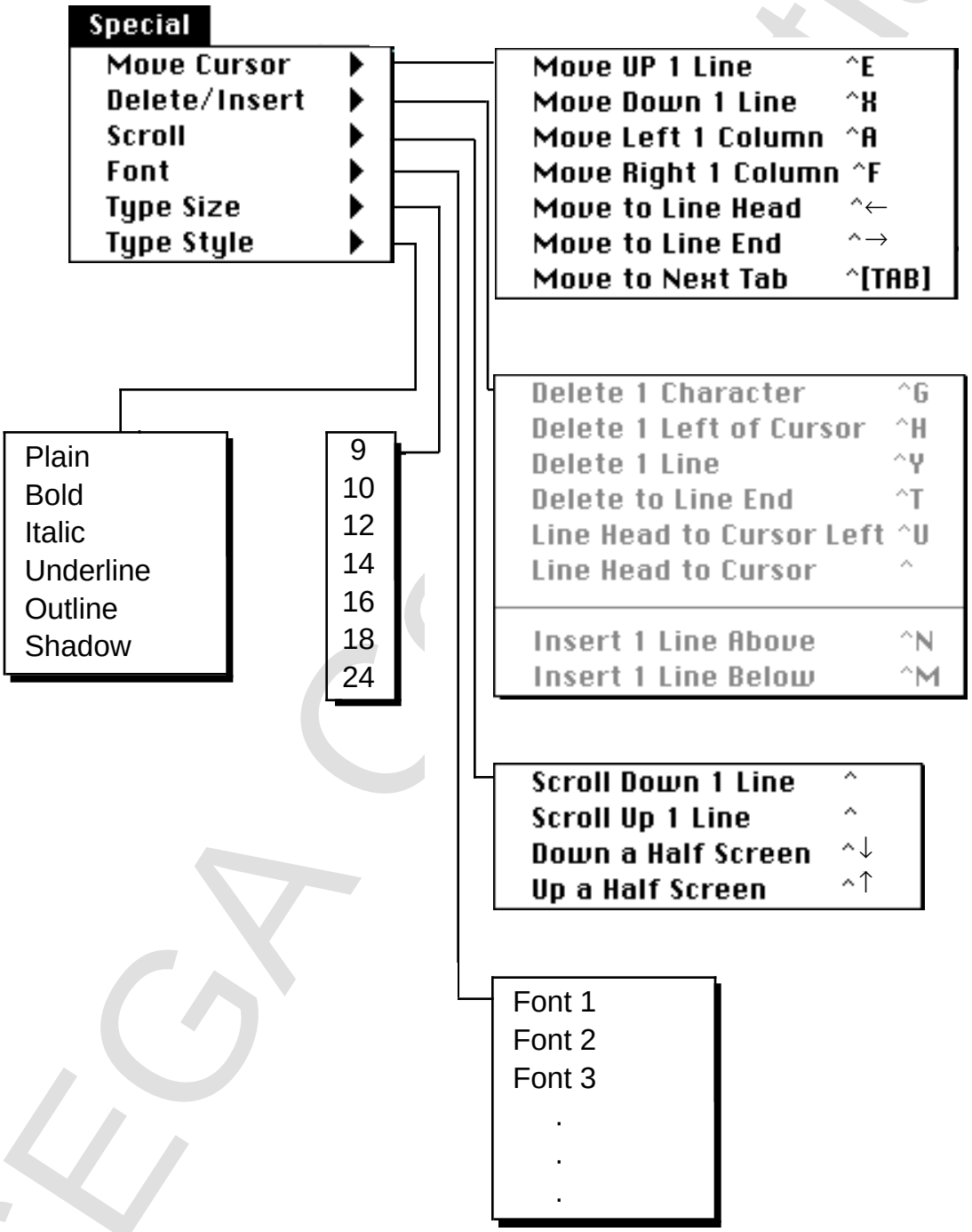


Selecting **sample1.asm** from among the three windows above activates the sample1.asm window as shown below.



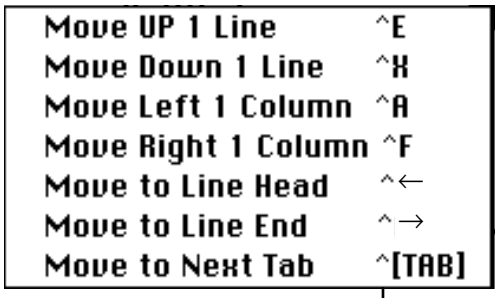
Special Menu

The Special menu lets you select items, using the mouse, that can be set by the **Set Shortcut Key** command in the Edit menu. It also includes text display change commands (font, size, style). Each command has a sub- menu.



Move Cursor

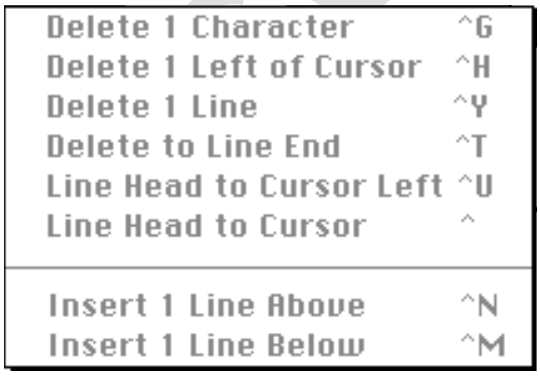
This is a set of commands for moving the cursor within the active text window. These commands can be operated by keys set by the **Set Shortcut Key** command in the Edit menu. Shortcut keys currently used are on the right side of the menu.



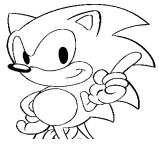
Current Shortcut key settings

Delete/Insert

This set of commands inserts and deletes characters within the active text window. These commands can be operated by keys set by the **Set Shortcut Key** command in the Edit menu. Shortcut keys currently used are on the right side of the menu.

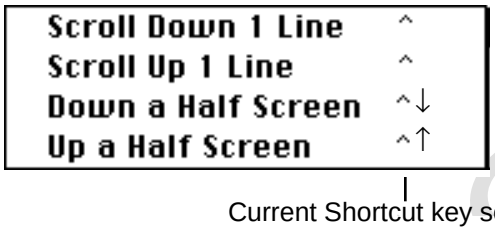


Current Shortcut key settings

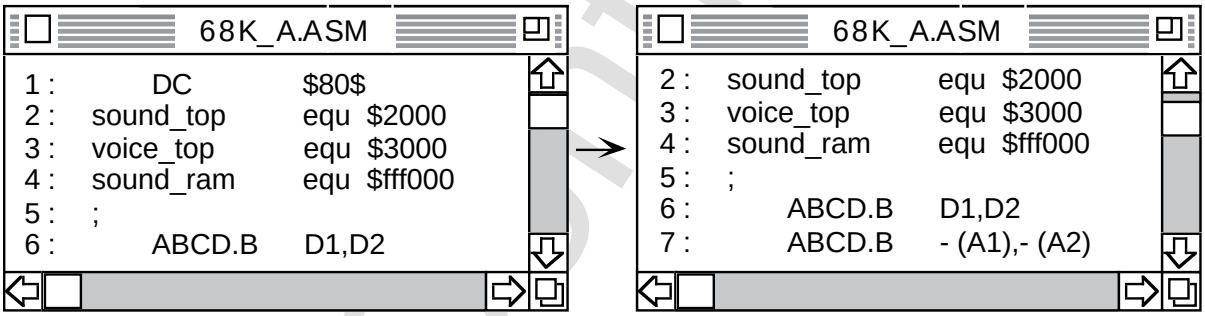


Scroll

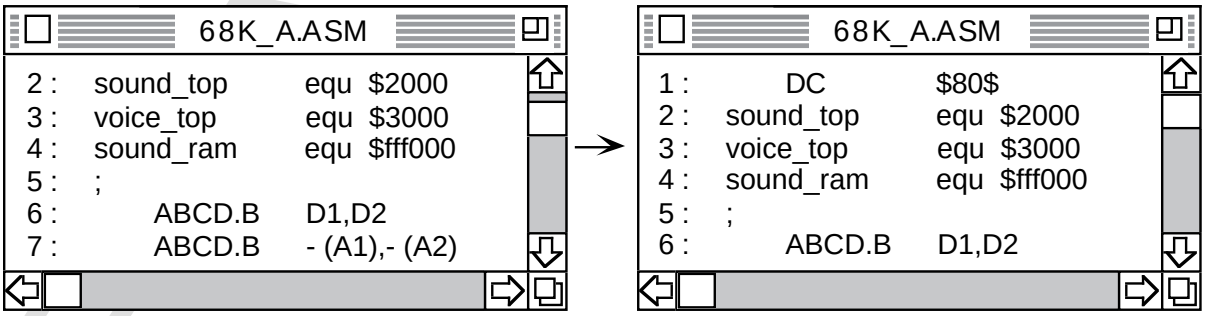
This is a set of commands for scrolling within the active text window. These commands can be operated by keys set by the **Set Shortcut Key** command in the Edit menu. Characters at the right side of the menu show the shortcut keys that are currently used.



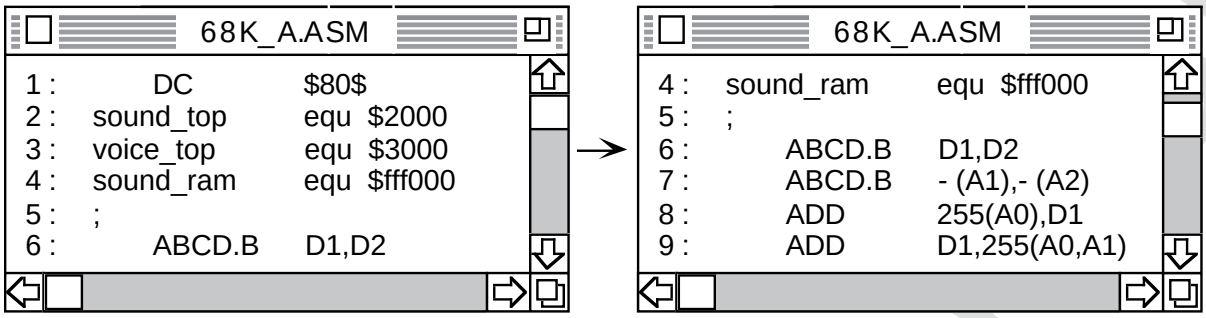
Scrolling down one line moves you down a single line:



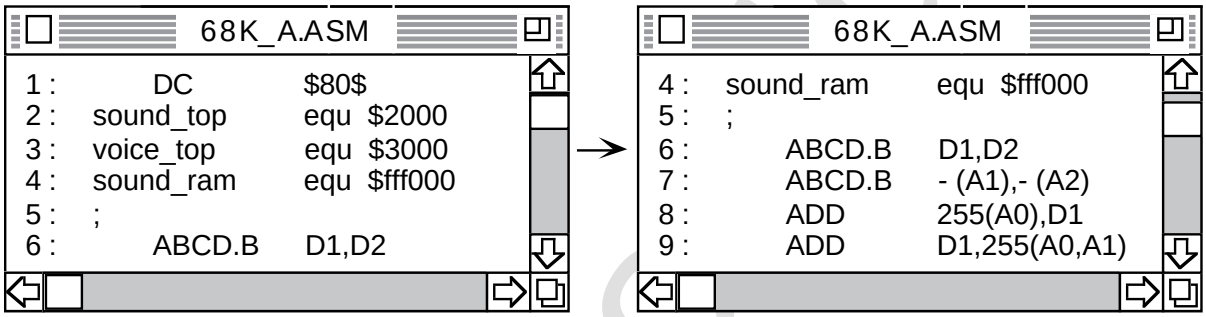
Scrolling up one line moves you up a single line:



Down Half Screen scrolls down half the screen, or approximately three rows:

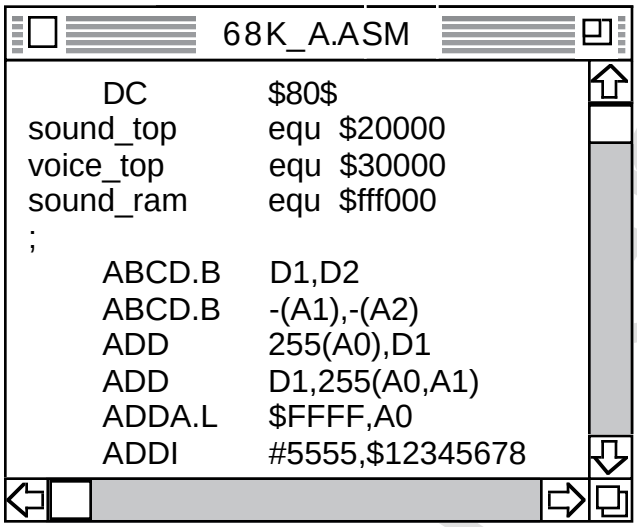


Up Half Screen scrolls up half the screen, or approximately three rows:

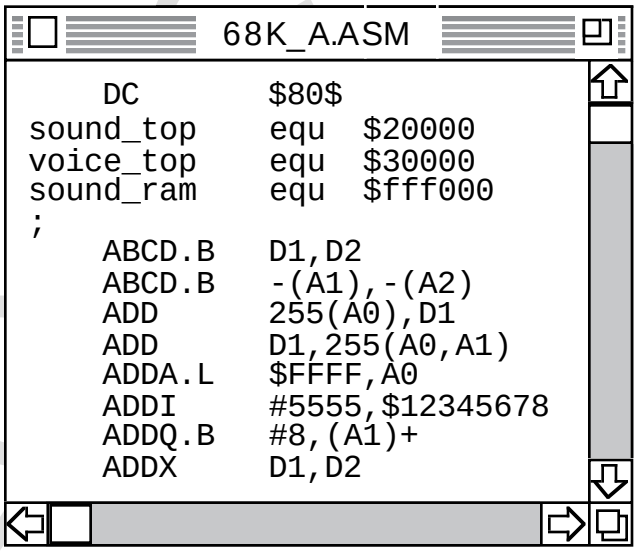


Font

Changes fonts in the active text window. Fonts that are added to the system folder when an application starts up are read and added to the menu. You can choose from among the following fonts to use.

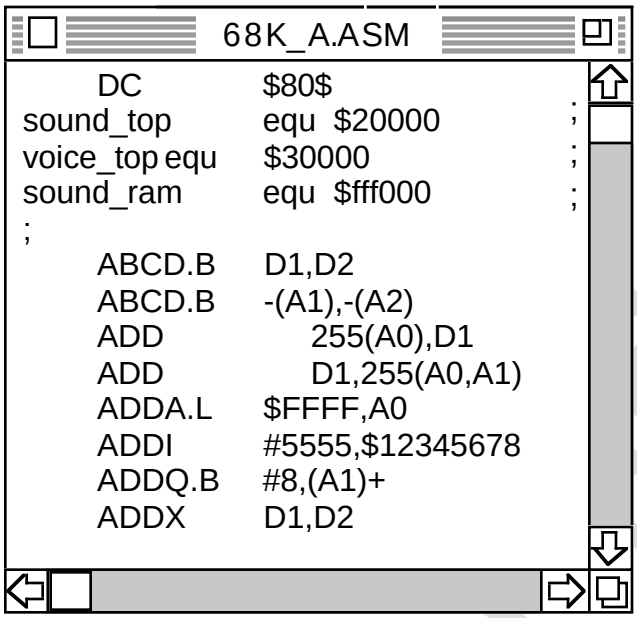


Font changed from Helvetica to Courier.

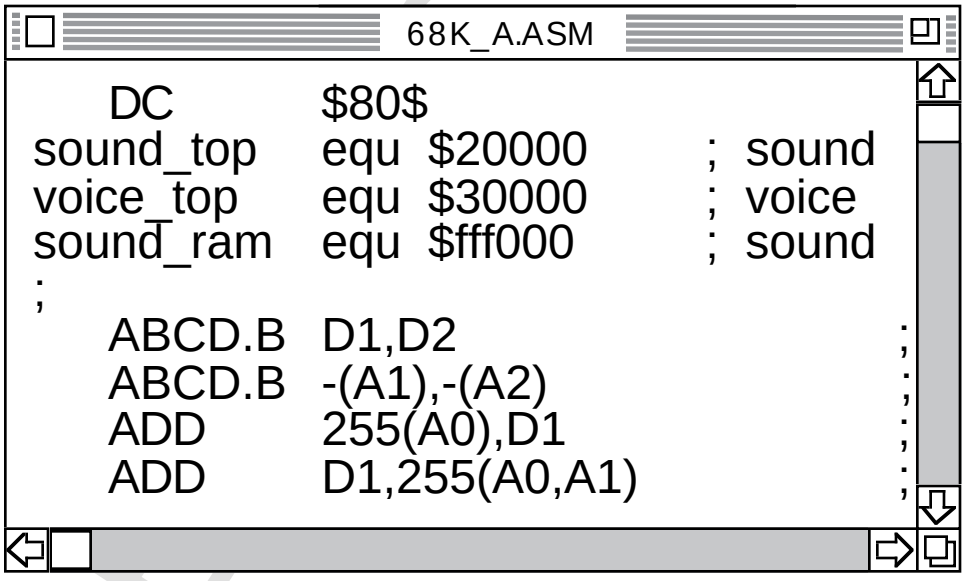


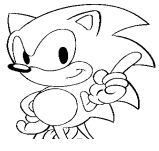
Type Size

Changes the character size in the active text window. Sizes include 9, 10, 12, 14, 16, 18, and 24. All characters within the screen will change to the size selected.



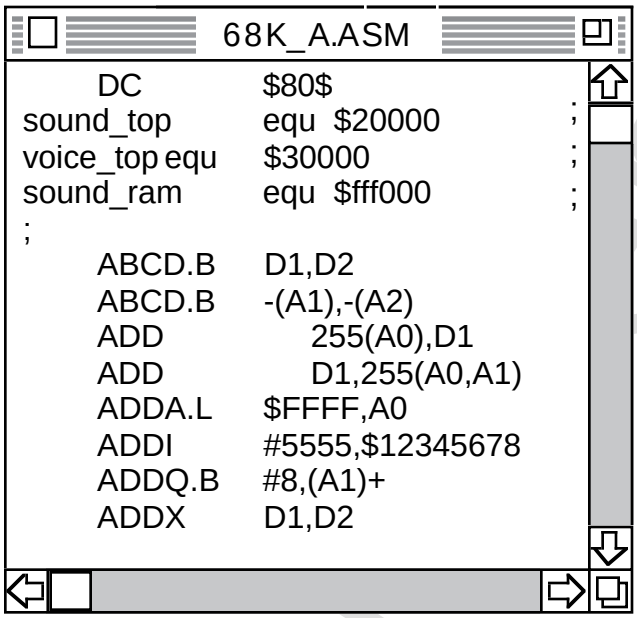
Font changed from 12 pt. to 18 pt.



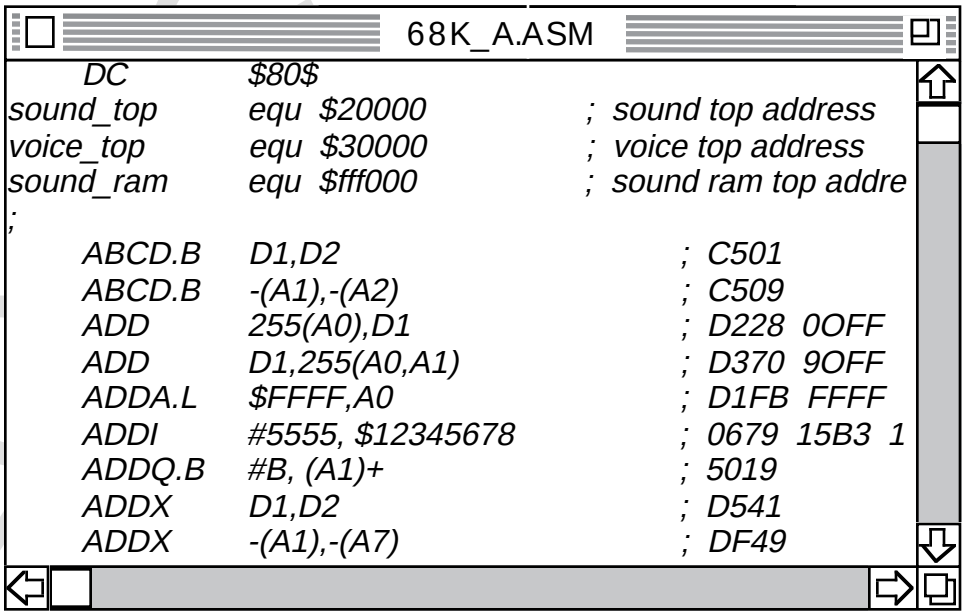


Type Style

Changes the character style in the active text window. Styles include plain (standard), bold, italic, underline, outline, and shadow. All characters within the screen will change to the style selected.



Style change from plain to italic.



6.0 Assembler Overview

About theAssembler Statement

Source 1 line of the assembler can be described by the following forms.

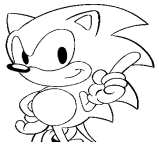
[Label]	<Command>	<Operand>	[Comments]

[ ] can be omitted but < > cannot be omitted if one or the other is described. The separating character of each element is a space or tab.

Label	When jumping from another line, the label is used as a reference by the name assigned to the description row. With a maximum of 32 character, you must begin at the first column. Characters that can be used are: a to z, A to Z, 0 to 9,?, @, _ (underbar) Numbers cannot be used in the lead character. * As special functions : : added to the end of the label means PUBLIC, # # added to the end of the label means EXTERN.
Comment	This is an explanation added to make a program more easily understood. It begins with a ; (semicolon) and ends with indentation. All characters in between will be treated as comments.

Example: An example with MC68000

```
START   MOVE.W   D0, D1   ; Contents in D0 sent to D1
  |       |        |            |
label  command  operand      comment
```

Expressions

Expressions can be used freely in commands and operands. Elements that make up the expressions are listed below.

- Symbols
- Operators
- Constants (numbers or characters)
- Location marks

Symbols

Symbols use character strings defined as labels within an expression.

Operators

Operators and their priority order are listed in the table below.

Operator	Meaning	Priority Order
*	multiplication	
/	division	1
MOD	remove remainder	
+	addition	2
-	subtraction	
SHR	shift left	3
SHL	shift right	
LAND	AND logic	
LOR	OR logic	4
LXOR	XOR logic	
EQ	equal	
NE	not equal	
LT	smaller than	5
LE	equal or smaller	
GT	larger than	
GE	equal or larger	
NOT	1 complement (Z80 only)	
NOT1	1 complement (68K only)	6
HIGH	high order byte	
LOW	low order byte	

Numerical Constants

Numerical constant is a number that is expressed by the following:

- Binary
- Octal
- Decimal
- Hexadecimal

Method	Format	Example
2	%bbbb bbbbB	%0111 1000B
8	qqQ qq0	017Q 110
10	dddd	10 255 1000
16	$hhhh hhhhH	$FF 1000H

b : 0 or 1
q : 0 to 7
d : 0 to 9
h : 0 to 9 A to F

Character Constants

Character constants are values, and considered ASCII code with one or two characters enclosed by quotation marks (‘).

Example:

‘ a’ 61H
‘a b’ 6162H

Location Marks

The location mark is expressed by “ * “ and is the location counter value at that time.

Example:

```
ABC     EQU     *
XXXX    * + 2
```


Attributes of an Expression

Symbols within the expression must belong to any of the following.

- Absolute (value is fixed)
- Relative (value is segment relative)
- External reference (symbol declared EXTERN)

Expressions, including these type symbols, must follow the rules below.

- All symbols used within a calculation other than addition and subtraction must be absolute.

- Addition
  One element must be absolute.
  Relative + absolute becomes relative.
  The elements of expressions containing external references must all be absolute.

- Subtraction
  Relative - absolute is relative.
  Subtraction by relative of equal segments are absolute.
  Other elements of expressions containing external references must all be absolute.

The rules above apply in each step while evaluating an expression.

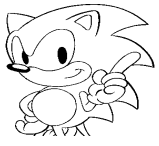
Example:

```
        CSEG
ABC     EQU     *
DEF     EQU     *
GHI     EQU     *
        DW      ABC+DEF-GHI . . .  ERR
        DW      ABC+(DEF-GHI)      . . .  OK
```

7.0 Assembler Pseudo-Instructions

Pseudo-instructions are shown in the table below.

Classification	Pseudo-instruction	Function
Link Control	ORG	Designates origin
	ASEG	Designates absolute segment.
	BSEG	Designates for user and extensive use segment.
	CSEG	Designates code and segment.
	DSEG	Designates data and segment.
	COMMON	Designates common block.
	END	Ends program.
	PUBLIC GLOBAL XDEF	Designates externally defined name
	EXTERN EXTRN XREF	Designates external reference name
Symbol Definition	EQU	Value allocation
	SET DEFL	Temporary value allocation
Data Definition	DB DEFB FDB	Defines bytes
	DW DEFW FDD	Defines words
	DC DEFM FCC	Defines characters
	DS DEFS RMB	Allocates memory
Macro Control	MACRO	Defines macro
	ENDM	Ends macro definition
	EXITM	Interrupts macro definition
	REPT	Repeats macro
	IRP	Continues macro
	IRPC	Character string macro
Conditional Assembler	IFDEF	Assemble if definition is complete
	IFNDEF	Assemble if there is no definition
	IFB	Assemble if operand is a space
	IFNB	Assemble if operand is not a space
	IFE	Assemble if expression value is 0
	IFNE	Assemble if expression value isn't 0
	IFIDN	Assemble if 2 character strings are equal.
		Assemble if 2 character strings differ.
	IFDIF	Assembles by reverse conditions of future IF.
	ELSE	End conditional assemble
	ENDIF	



Classification	Pseudo-instruction	Function
Output Control	.LIST .XLIST .MACRO .XMACRO .IF .XIF	Outputs list Discontinues output list Outputs macro expansion Discontinues output macro expansion. Outputs condition skip Discontinues output condition skip
Other	INCLUDE INCL TITLE PAGE RADIX	Read file Designates list title Designates form feed or number of rows in one page. Designates radix

ORG

```
ORG   <expression>
```

Sets the <expression> value at the location and counter. The code to be manufactured is assigned from the address of that value. <expression> must be an absolute expression already determined by value.

Example:

address	code		
		org	8000H
00008000	12345678	dl	$12345678
00008004	30388000	move	$8000.W,d0

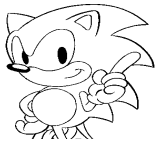
BSEG, CSEG, DSEG, COMMON

```
BSEG
CSEG
DSEG
COMMON
```

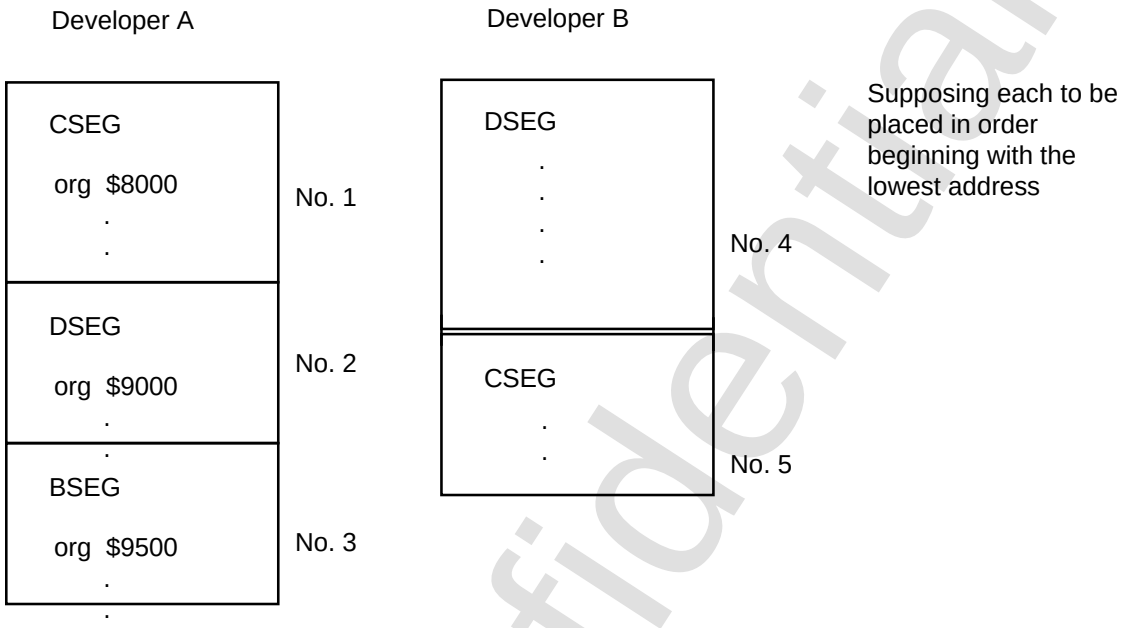
These commands set the code and relative address in memory into the location and counter. After a command, the location and counter values become the final address of each segment until that location value is not changed by ORG. Also, when activated, the location and counter value of each segment are initialized by the value set by the option.

Example:

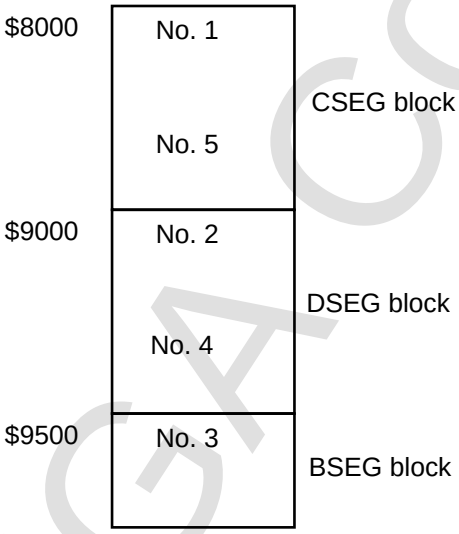
```
DSEG
dw      $0001
dw      $0002
dw      $0003
CSEG
move        d0,$8000
jmp         start
  .
  .
  .
```

When all modules are created individually:



If these two modules are linked:



As shown above, even modules that were created separately can be grouped together when linking parts that resemble each other. Generally, CSEG is used exclusively for programs and DSEG is used exclusively for data. BSEG is peculiar to this program (SDSS), and is an application segment that can be used freely by the user.

END

Shows the end of the program. Any program after END is ignored.

Example:

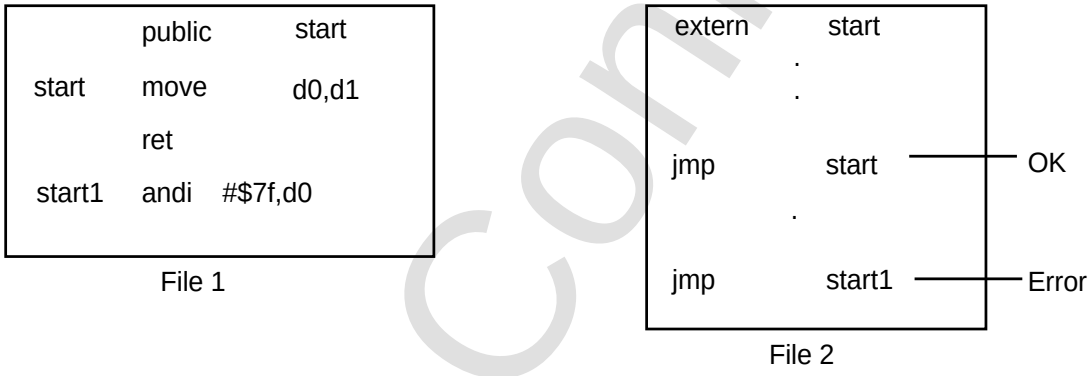
```
        org     $8000
        add     d0,d1
                .
                .
                .
        END
```

PUBLIC, GLOBAL

```
PUBLIC      <symbol>{,<symbol>}
GLOBAL      <symbol>{,<symbol>}
```

Symbols that are declared PUBLIC can refer to symbols within other programs. The EXTERN declarative is necessary for using modules.

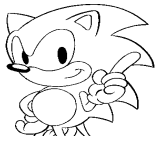
Example:



EXTERN, EXTRN

```
EXTERN      <symbol>{,<symbol>}
EXTRN       <symbol>{,<symbol>}
```

To refer to symbols defined by other modules, an EXTERN declarative is required. (See PUBLIC, GLOBAL example).



EQU

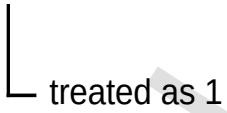
```
<label>   EQU   <expression>
```

Assigns <expression> values to labels.

A label defined by an EQU pseudo-instruction cannot be redefined by parts of other programs. Also, <expression> can not use an outside reference or previously referenced symbol.

Example:

```
abc          equ    $01
move.w       #abc,d0
```



SET, DEFL

```
<label>   [:] SET   <expression>      (valid only for 68K)
<label>   [:] DEFL  <expression>      (valid only for Z80)
```

Assigns <expression> values to label.

SET pseudo-instructions, different from EQU, can redefine identical labels indefinitely. Limitations on <expression> are the same as with EQU.

Example:

```
abc          set    $01
move.w       #abc,d0 ——— abc in this line is handle as 1
abc          set    $02
move.w       #abc,d0 ——— abc in this line is handle as 2
```

DB, DEFB, FCB

```
DB      <expression>      {,<expression>}
DEFB    <expression>      {,<expression>}
FCB     <expression>      {,<expression>}
```

The value of 8 bits corresponding to the operand <expression> is placed in memory. More than one operand can be used, each separated by a comma. Omitting the area between commas has the same meaning as designating 0.

Example:

address	code		
		org	8000H
00008000	12345678	db	$12,$34,$56,$78
00008004	01020001	db	1,2,,1

DW, DEFW, FDB

```
DW       <expression>     {,<expression>}
DEFW     <expression>     {,<expression>}
FDB      <expression>     {,<expression>}
```

The value of 16 bits corresponding to the operand <expression> is placed in memory. More than one operand can be used, each seperated by a comma. Omitting the area between commas has the same meaning as designating 0.

Example:

address	code		
		org	8000H
00008000	00010002	dw	1,2,,1
	00000001		
00008008	1234	defw	$1234

DL

```
DL   <expression>   {,<expression>}
```

The value of 32 bits corresponding to the operand <expression> is placed in memory. More than one operand can be used, each seperated by a comma. Omitting the area between commas has the same meaning as designating 0.

Example:

address	code		
		org	8000H
000080000	12345678	dl	$12345678
00008004	1234	dw	$1234

DC, DEFM, FCC

```
DC      /<ASCII character string>/
DEFM    /<ASCII character string>/
FCC     /<ASCII character string>/
```

The character string is put into the ASCII code and that value is placed in memory. The symbol that encloses the character string does not have to be a slash (/), but the last symbol must be the same as the first symbol.

Example:

address	code		
		org	8000H
00008000	61626364	dc	/abcdef/
00008004	6566		

DS, DEFS, RMB

```
DS      <expression>
DEFS    <expression>
RMB     <expression>
```

The amount of bytes designated by <expression> is maintained in memory. <expression> must be an absolute expression with an already decided value.

Example:

address	code		
		org	8000H
00008000	61626364	dc	/abcdef/
00008004	6566		
00008006		ds	100
0000806A	C100	abcd	d0,d0

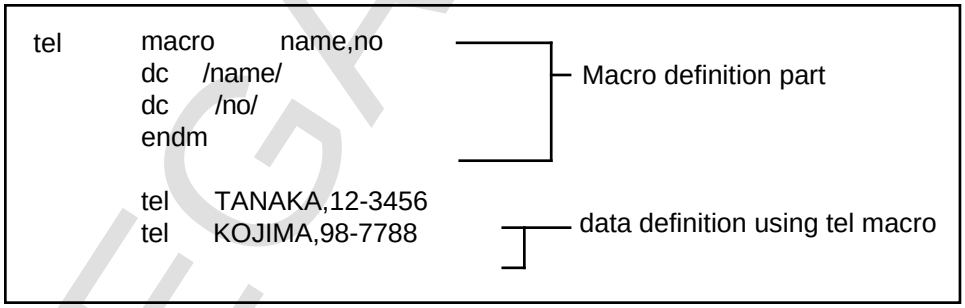
MACRO ~ ENDM

```
<Macro Name>     MACRO      argument...,
                    .

                 ENDM
```

Macro definition is performed between MACRO and ENDM.
Limitations on <Macro Name> are the same as normal labels.

Example: When you want to define a name and telephone number as character string data.



The above data definitions are developed as shown on following page.

```
                    tel             TANAKA,12-3456
+00000000           54414E41        dc      /TANAKA/
+00000004           4B41
+00000006           31322D33        dc      /12-3456/
+0000000A           343536
                                    tel     KOJIMA,98-7788
+0000000D           4B4F4A49        dc      /KOJIMA/
+00000011           4D41
+00000013           39382D37        dc      /98-7788
+00000017           373838
```

Using macro definitions gives a program that is more advanced than using a group of commands held by the original MPU (micro processing unit). For example, data definitions of telephone numbers are:

```
68K Command direct                  Macro Use
dc    /TANAKA/                      tel   TANAKA,12-3456
dc    /12-3456/                     tel   KOJIMA,98-7788
dc    /KOJIMA/
dc    /98-7788/
```

Here, the descriptor becomes simpler and easier to understand.

EXITM

```
EXITM
```

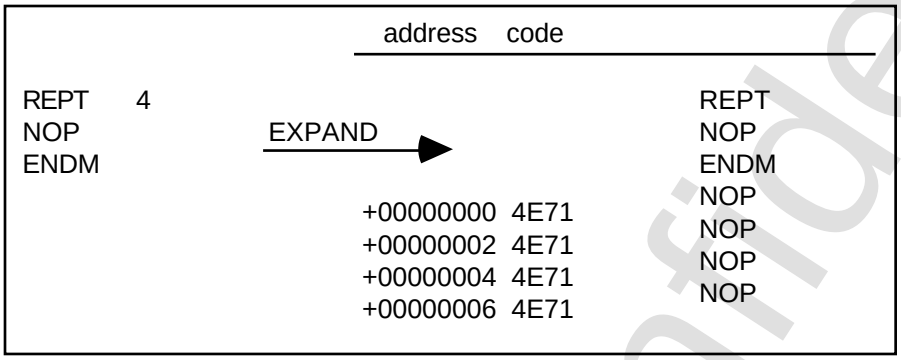
EXITM pseudo-instructions are used to force the end of MACRO, REPT, IRP, IRPC. If EXITM is executed, macro expansion is immediately stopped and moves to the next line.

REPT ~ ENDM

```
<Macro Name>   REPT   <expression>
                 .
               ENDM
```

REPT expands the part from REPT to ENDM only the number of times that the absolute expression is displayed by <expression>.

Example:

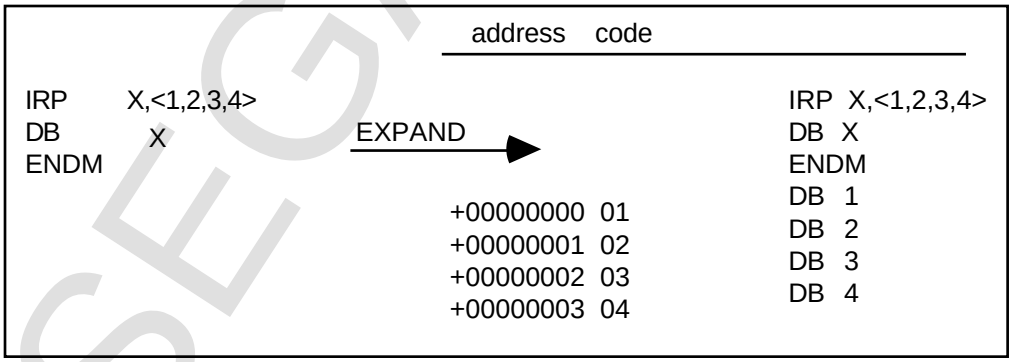


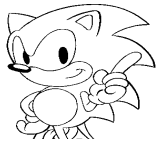
IRP ~ ENDM

```
<Macro Name>   IRP   <dummy>,<argument list>
                 .
               ENDM
```

IRP expands until the argument list has been replaced by dummies.

Example:



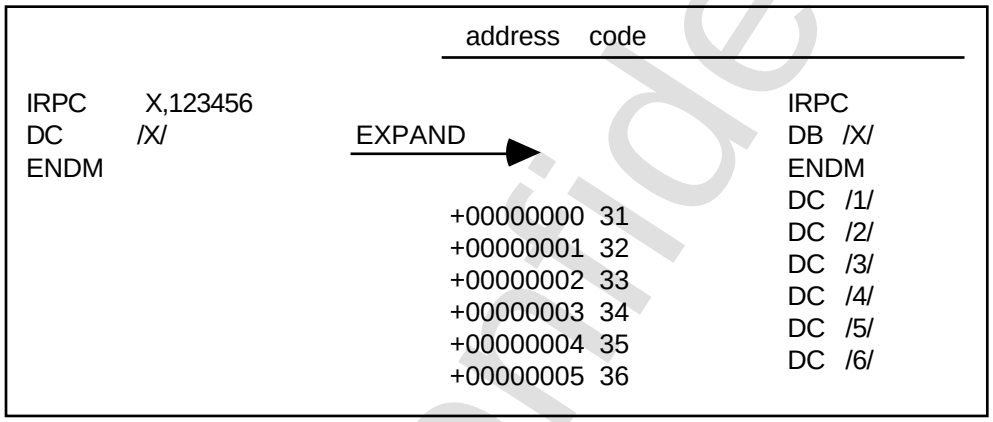


IRP ~ ENDM

```
<Macro Name>   IRPC   <dummy>,<argument list>
                 .
               ENDM
```

IRPC has as an argument a character string, and expands by replacing each character of the part until ENDM.

Example:

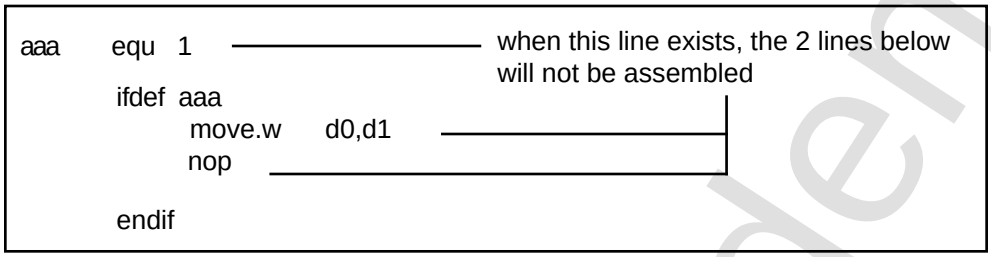


IFDEF ~ ENDIF

```
IFDEF    <symbol>
ENDIF
```

If <symbol> is predefined, assemble continues until ELSE or ENDIF.

Example:

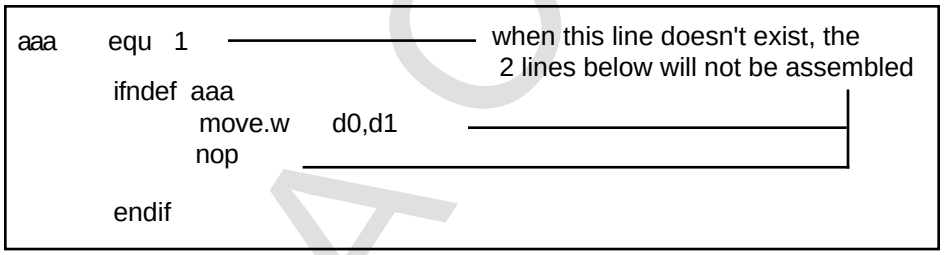


IFNDEF ~ ENDIF

```
IFNDEF   <symbol>
ENDIF
```

If <symbol> is undefined, assemble continues until ELSE or ENDIF.
IFDEF ~ ENDIF has a reverse function.

Example:



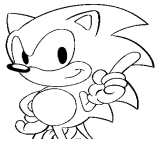
IFB ~ ENDIF

```
IFB  <character string>
```

If there are no characters between "<" and ">" assemble continues until ELSE or ENDIF.

Example:

```
ifb   <>
      move.w   d0,d1   ——— can assemble
endif
```



IFNB ~ ENDIF

```
IFNB  <character string>
ENDIF
```

If there are characters between "<" and ">" assemble continues until ELSE or ENDIF. IFB ~ ENDIF is a reverse function.

Example:

```
ifnb <abc>
     move.w   d0,d1 _______– can assemble
endif
```

IFE ~ ENDIF

```
IFE          <expression>
```

If the value of <expression> is 0, assemble continues until ELSE or ENDIF.

Example:

```
bbb  equ  1
ife  bbb-1
     move.w   d0,d1 _______– can assemble
endif
```

IF ~ ENDIF, IFNE ~ ENDIF

```
IF     <expression>      or      IFNE     <expression>
```

If <expression> has a value other than 0, assemble continues until ELSE or ENDIF. IFE ~ ENDIF has a reverse function.

Example:

```
bbb  equ  1
if   bbb-1
     move.w   d0,d1 _______– can not assemble
endif
```

IFIDN ~ ENDIF

```
IFIDN     <character string>,<character string>
```

If two character strings are equal, assemble continues until ELSE or ENDIF. Character strings must be enclosed by "<" and ">".

Example:

```
ifidn  <same>,<same>
        move.w     d0,d1 _______  can assemble
endif
```

IFDIF ~ ENDIF

```
IFDIF     <character string>,<character string>
```

If two character strings are different, assemble continues until ELSE or ENDIF. Character strings must be enclosed by "<" and ">". IFIDN ~ ENDIF has a reverse function.

Example:

```
ifdif  <same>,<same>
        move.w     d0,d1              can not assemble
endif              _____
```

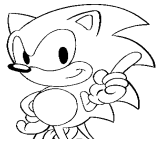
ELSE, ENDIF

```
ELSE
ENDIF
```

ELSE performs assemble or skips according to conditions opposite of the leading conditional assemble command. Used in the form of IF** ~ ELSE ~ ENDIF. ENDIF shows the end of conditional assemble.

Example:

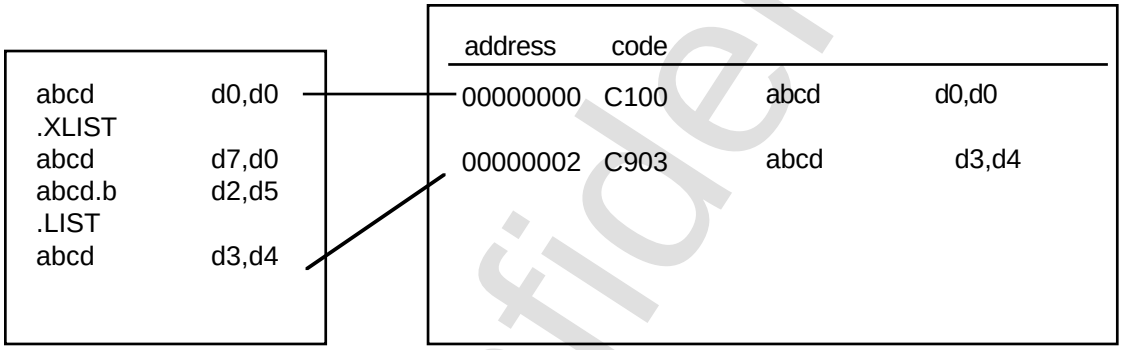
```
ifb   <>
      move.w      d0,d1          can assemble
else
      move.w      d0,d1          can not assemble
endif
```

.LIST, .XLIST

```
.LIST
.XLIST
```

.LIST is the condition allowing list output.
.XLIST is the condition not allowing list output. .LIST is the default. When the list output switch is off in the project, list output will not be performed unconditionally.

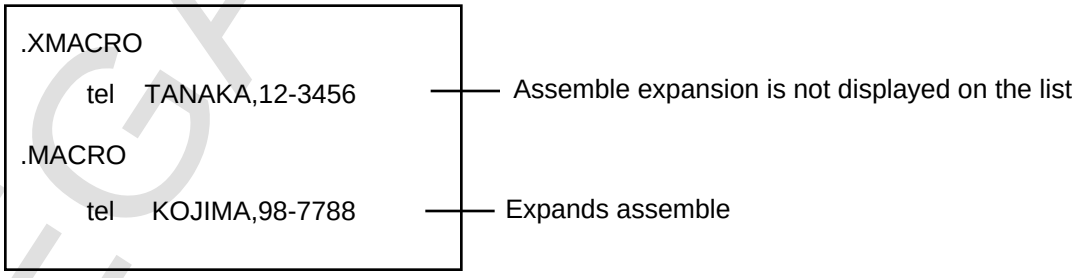


Source File List File

.MACRO, .XMACRO

```
.MACRO          can output macro expansion list
.XMACRO         can not output macro expansion list
```

Default is possible. But when the list output switch is off in the project, a list cannot be output unconditionally.



Source File

.IF, .XIF

`.IF` Able to output a list of lines skipped by the conditional assemble function
`.XIF` Not able to output a list of lines skipped by the conditional assemble function

In the default setting the list can be output. But when the project list output switch is off in the project, a list can not be output unconditionally.

Example:

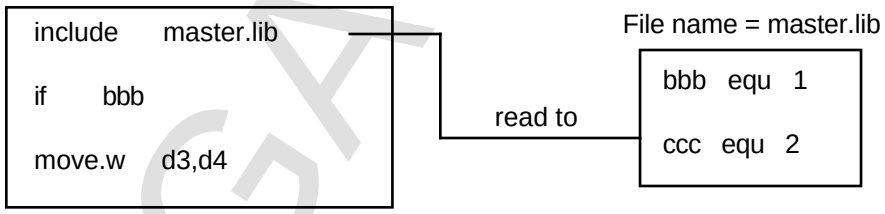
```
bbb   equ 1
  .XIF
  if      bbb
          move.w d0,d1  ____ can assemble and output a list
  else
          move.w d0,d1  ____ can not assemble and output a list
endif
```

INCLUDE, INCL

```
INCLUDE    <file name>
INCL       <file name>
```

Reads and assembles files displayed by <file name>.
INCLUDE permits nesting of up to 10 levels.

Example:



Source File

TITLE

```
TITLE   <character string>
```

Sets the title of the list file.

Example:

If TITLE TEST
are described in source file ABCD.ASM, in ABCD.LST created after assemble:

SDSS (MC68000) Cross Macro Assembler - Macintosh - Version 1.0 30 Nov. 1993

<ABCD.ASM> TEST

2 lines containing title characters in the location where the page is divided

RADIX

```
RADIX   <expression>
```

Designates the radix. The radix is designated to be 10 at the time of start up. With <expression> as an absolute expression, the value must be either 2, 8, 10, or 16.

Example:

```
address     code
                        radix       16
00000000    1234        dw    1234
                        radix       10
00000002    04D2        dw    1234
```

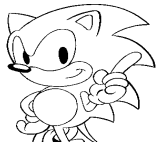
NAME, NAM

```
NAM
NAME
```

Designates the module name, which does not have to be designated. If it is designated when linking, it will be displayed in the map file.

8.0 List of Assembler Linker Error Messages

Output Message	Remarks
Assemble error	There is an error in the assemble file The error is display along with the file name
Line invalid	Line is not correct
No source file specified	No source file was specified
Could not open file	Source file could not be opened
Could not create file	File could not be created
Too many nesting files	File nesting is too deep (including nest)
Too many nesting macros	Macro nesting is too deep
Label buffer full	Too many labels
Macro buffer full	Too many macros
Unterminated conditional	control such as # if are not closed
Unterminated macro	Macro is not closed
Not enough memory for SDSS	Not enough memory to start up SDSS
Many errors	There are too many errors
Relative branch out of range	Long distance between relative jumps
Constant was expected	No constant at location where it should be
Reference to multi defined	Performs multiple definitions
Extra characters on line	Unreadable characters exist
Already had ELSE clause	There is no IF that corresponds to ELSE
Illegal expression	Illegal expression

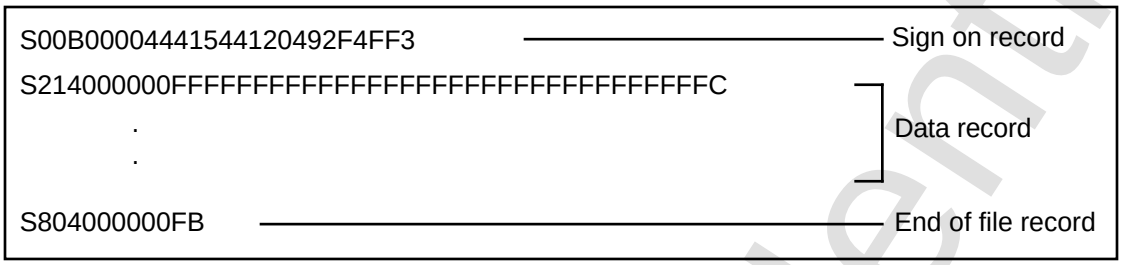


Output Message	Remarks
Label was expected	Label was not at required location
Symbol is multi defined	Duplicate symbols
Operand was expected	No operand
Permanent label was expected	No permanent label
Symbol is reserved word	Reserved word was used
Unexpected end of strings	String ends at an unexpected location
Too complex expression	Expression used is too complex
Symbol not defined	Undefined symbol
Value is out of range	Value exceeds range
Symbol already external	External symbol was used as local
Syntax error	Syntax error
Division by 0	Divides by 0
Unexpected end of file	End of file can not be found
BAD D8 RELATIVE	Exceeded the range of the 8 bit relative address
BAD D16 RELATIVE	Exceeded the range of the 16 bit relative address

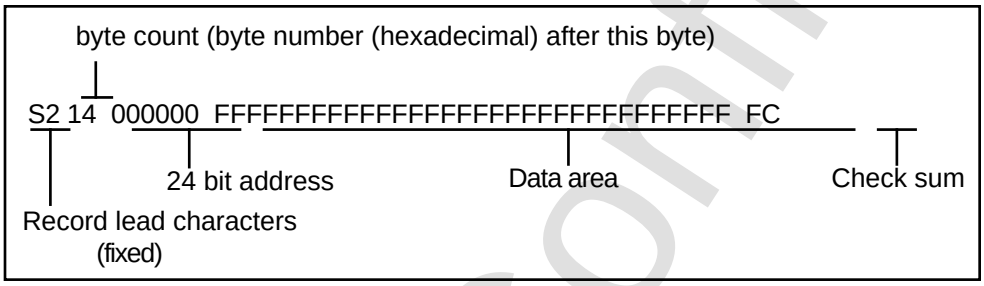
9.0 Load Module Output Format

Motorola S28

The following is one output format defined by Motorolla that outputs a 24-bit address code.



Contents of a data record are shown below.

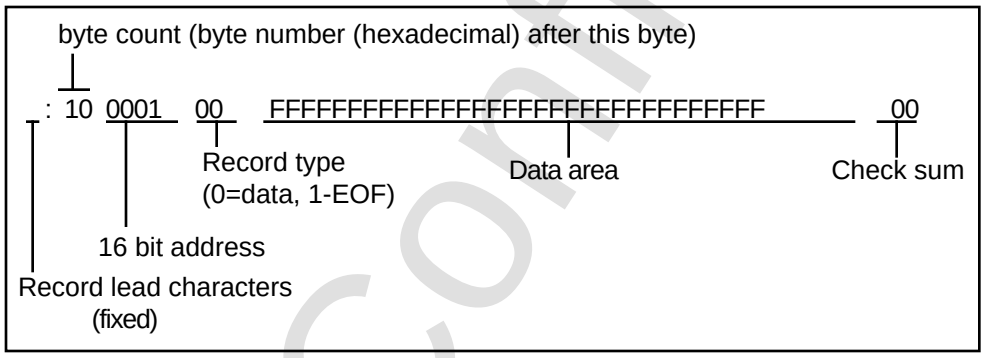


The check sum is a complement of single binary addition as far as the byte immediately before check sum within the code that contains the byte count, address, and data.

Intel HEX

The Intel format is one output format; it's output is in a 16 bit address code. Each record begins with a colon (:) and then continues with the byte count. After the byte count is the 4 digit address (in hexadecimal), and the record type. Next is the line of data, and finally the check sum.

Contents of a data record are shown below.



The check sum is a complement of double binary addition as far as the byte immediately before check sum within the code that contains the byte count, address, record type, and data.

Binary Format Output

This is a format that outputs the linked hexadecimal data code just as it is.
Data can only be output without having to output information such as an address.