2023

ASMEDIT X16 (v2)

REFERENCE MANUAL

GEOFFREY WAREING

Contents

1. INTRODUCTION

1.1 DESCRIPTION

ASMEDIT X16 is a menu driven '65C02' assembler and editor for the Commander X16. Source code can be created, saved, recalled, edited and assembled with listings. It also has a file maintenance facility to organise code on disk. The software, originally developed for the Commodore CBMII Series, has been adapted and ported to the Commander X16 environment.

1.2 HARDWARE REQUIREMENTS

Commander X16 (512k)

Compatible printer (optional)

1.3 LOADING THE PROGRAM

ASMEDIT X16 can be loaded with command LOAD "ASMEDIT.PRG" (with the current directory set to ASMEDIT) followed by the 'RUN' command. ASMEDIT.PRG will load the following binaries before entering the PRIMARY COMMANDS menu and be ready for use: -

XAM.BIN	Menu Control
XAE.BIN	Menu Functions
X0F.BIN	Machine Specific Functions
XAI.BIN	Default Input Data
XHP.BIN	Help Text
XAL.BIN	Browse Listing

**** It is suggested that HANDLING CODE in the appendices is read so as to gain a quick insight into the way in which source code is held by the program and the ways in which code may be structured.*

**** To gain a quick insight into ASMEDIT's manner of operation it is suggested that the example in APPENDIX B is now followed using the Commander X16.*

**** See the ASMEDIT 'Quick Start' video on YouTube:* https://youtu.be/HTeLqSqDS84

2. MENUS

2.1 DESCRIPTION

A menu is simply a screen display which either:

a) Lists options for the User to select a function (e.g., ‘LOAD CODE’)
b) Presents parameters required by a function (e.g., ‘CODE NAME’)

2.2 INPUT AREAS

Menus which require parameter input have what are termed 'Input Areas'. Input areas are displayed on the menu in reverse video. To the left of each area is a narrative which identifies what value the area should contain. Many input areas contain default values. Values entered in an input area are remembered between the exit and re-entry of a menu.

2.3 NAVIGATION

When a menu is first displayed a flashing cursor is place in the leftmost position of the first input area. Movement of the cursor across input areas is achieved with the vertical cursor keys. Movement within an area is achieved with the horizontal cursor keys. Values are entered by simply pressing the appropriate keys. The function keys available are as follows: -

• [INS/DEL]	to insert and delete characters respectively.
• [left arrow]	to clear characters to the right of the cursor inclusive.
• [RETURN]	to validate the input area values and invoke the menu function.
• [ESC]	to terminate the function before or during the execution.

2.4 INPUT VALIDATION

If an invalid value is input a message INVALID INPUT will be displayed at the bottom lefthand side of the screen and the input area{s) in error high-lighted with '?'.

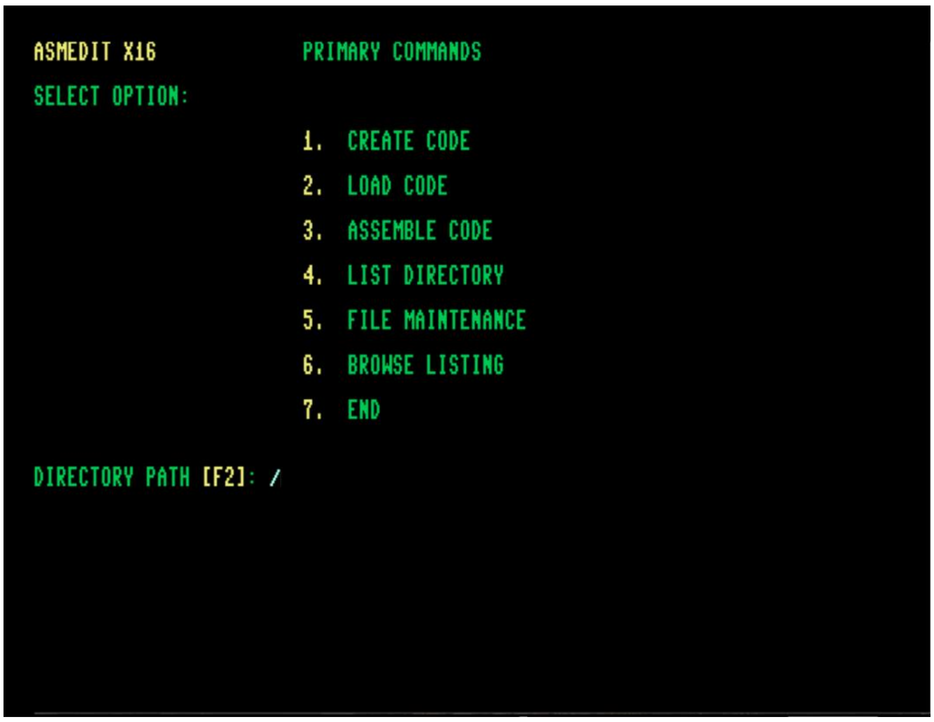
If an error occurs during the process an error message will be displayed at the foot of the menu screen. If a DOS error occurs the DOS message will be displayed as well. (See *MENU ERROR MESSAGES* for a list of messages which may occur).

2.4 FUNCTION

Once [RETURN] has been keyed, with valid values in the input areas, the menu’s function will be performed.

3. PRIMARY COMMANDS

The first menu presented on entry to ASMEDIT is the PRIMARY COMMANDS menu.



```
ASMEDIT X16             PRIMARY COMMANDS
SELECT OPTION:

                        1.  CREATE CODE
                        2.  LOAD CODE
                        3.  ASSEMBLE CODE
                        4.  LIST DIRECTORY
                        5.  FILE MAINTENANCE
                        6.  BROWSE LISTING
                        7.  END

DIRECTORY PATH [F2]: /
```

The required function is selected by keying the indicated number. e.g., Press [2] to select the LOAD CODE function.

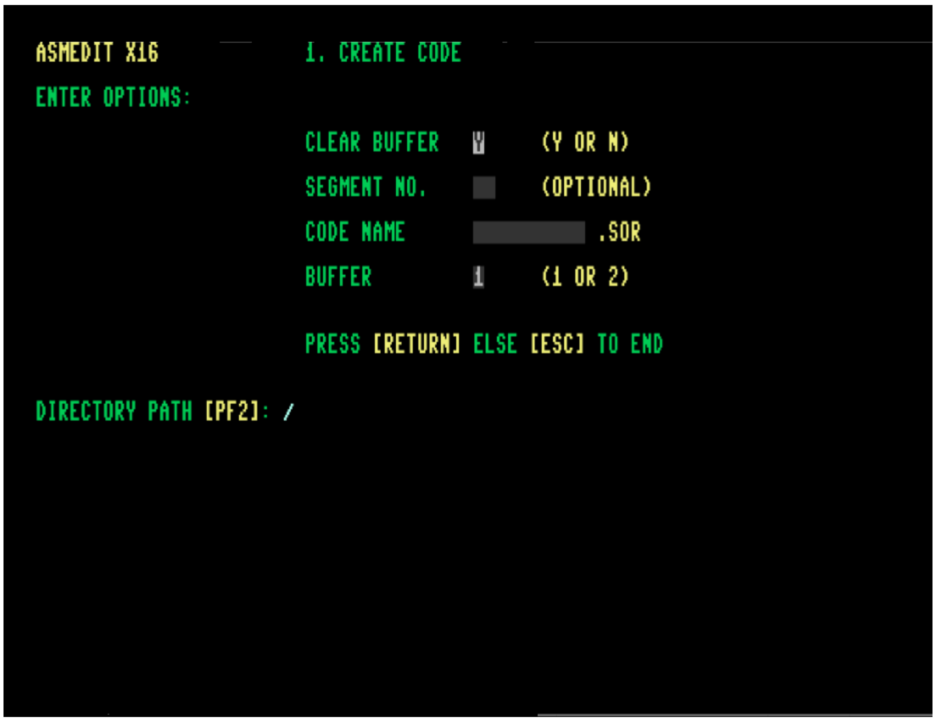
Note: The current DIRECTORY PATH is displayed on each menu ('/' means the root). Pressing [F2] allows the user to change the current directory at any point.

A detailed description of each function follows.

3.1 CREATE CODE (MENU 1)

This function is selected when it is required to: -

- Clear the Edit buffer prior to entering new source code, which may or may not be stored on disk later.
- Create new source code to be stored on disk from new or existing code already held in the Edit *BUFFER*.



```
ASMEDIT X16          1. CREATE CODE
ENTER OPTIONS:

                     CLEAR BUFFER   Y    (Y OR N)
                     SEGMENT NO.         (OPTIONAL)
                     CODE NAME              .SOR
                     BUFFER         1    (1 OR 2)

                     PRESS [RETURN] ELSE [ESC] TO END

DIRECTORY PATH [PF2]: /
```

The EDITOR is invoked to create, browse or amend code and is described fully in *4. THE* EDITOR section. The menu displays the following input areas:

CLEAR BUFFER (Y OR N)

This input area is fairly self-explanatory. The term buffer simply refers to the area where code is to be held when it is being keyed in and manipulated. The EDITOR is therefore asking whether any code currently in this area is to be erased (type 'Y') or whether it is to form part of the new code (type 'N').

SEGMENT NO. & CODE NAME

SEGMENT NO. and CODE NAME are appended together to form a file name representing the segment on disk. After editing, code will be filed on disk with this file name. SEGMENT NO. is optional but if entered must be a number from 1 to 99. The first segment number of the program must always be 1 and subsequent segments incremented by 1 in sequence.

BUFFER (1 OR 2)

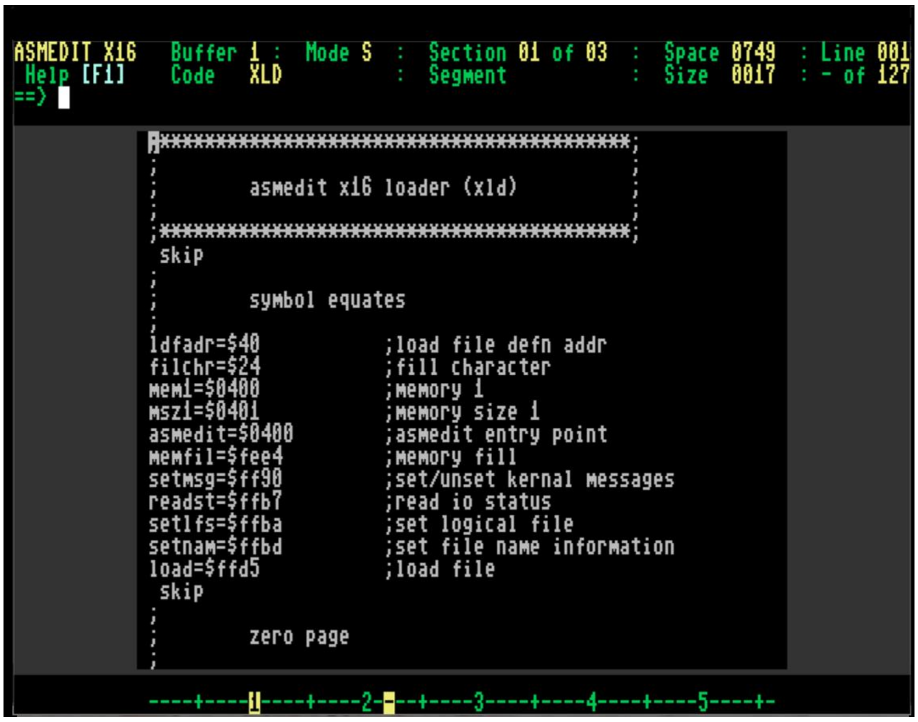
There are two buffers in which source code may be held. The value entered should be 1 or 2 to specify which buffer is to be used.

Press [RETURN] to execute function otherwise [ESC] to return to the PRIMARY COMMANDS menu.

Note: The current DIRECTORY PATH is displayed ('/' means the root). Pressing [F2] allows the user to change the current directory at any point.

3.2 LOAD CODE (MENU 2)

This function is selected when required to edit existing source code held on disk. Loading from disk however, occurs only if the source code required is not contained in the Edit Buffer already.



```
ASMEDIT X16   Buffer 1 :  Mode S  :  Section 01 of 03  :  Space 0749  : Line 001
 Help [F1]    Code   XLD          :  Segment           :  Size  0017  : - of 127
==>

;***************************************;
;                                       ;
;        asmedit x16 loader (xld)       ;
;                                       ;
;***************************************;
 skip
;
;        symbol equates
;
ldfadr=$40             ;load file defn addr
filchr=$24             ;fill character
mem1=$0400             ;memory 1
msz1=$0401             ;memory size 1
asmedit=$0400          ;asmedit entry point
memfil=$fee4           ;memory fill
setmsg=$ff90           ;set/unset kernal messages
readst=$ffb7           ;read io status
setlfs=$ffba           ;set logical file
setnam=$ffbd           ;set file name information
load=$ffd5             ;load file
 skip
;
;        zero page
;

----+----1----+----2----+----3----+----4----+----5----+-
```

The EDITOR is invoked to create, browse or amend code and is described fully in 4. THE EDITOR section. The menu displays the following input areas:

SEC TION NO.

This input area specifies which SEGMENT No. is to be displayed on entry to the Editor. If the section specified is not numeric or greater than the number of sections in the current segment the input area will be flagged and the function will fail.

SEGMENT NO. & CODE NAME

SEGMENT NO., CODE NAME and '.SOR' are appended together to form a file name which represents a SEGMENT on disk. The current DIRECTORY will be searched for the file name constructed from the SEGMENT and CODE NAME entered. If found the code file will be loaded into the Editor's buffer. Later the amended code may be filed on disk under this file name overwriting the previous copy. SEGMENT NO. is optional but if entered must be a number from 1 to 99, beginning with 1.

'?', when specified within the SEGMENT or CODE NAME input area, has a special meaning. Instead of attempting to load code with '?' in the file name ASMEDIT searches the current disk directory and displays all files which match the characters given between the '?'s specified. The user then has the option of scrolling up or down and selecting the file required. Ref. WILD CARD FEATURE for useful examples of this feature.

BUFFER (1 OR 2)

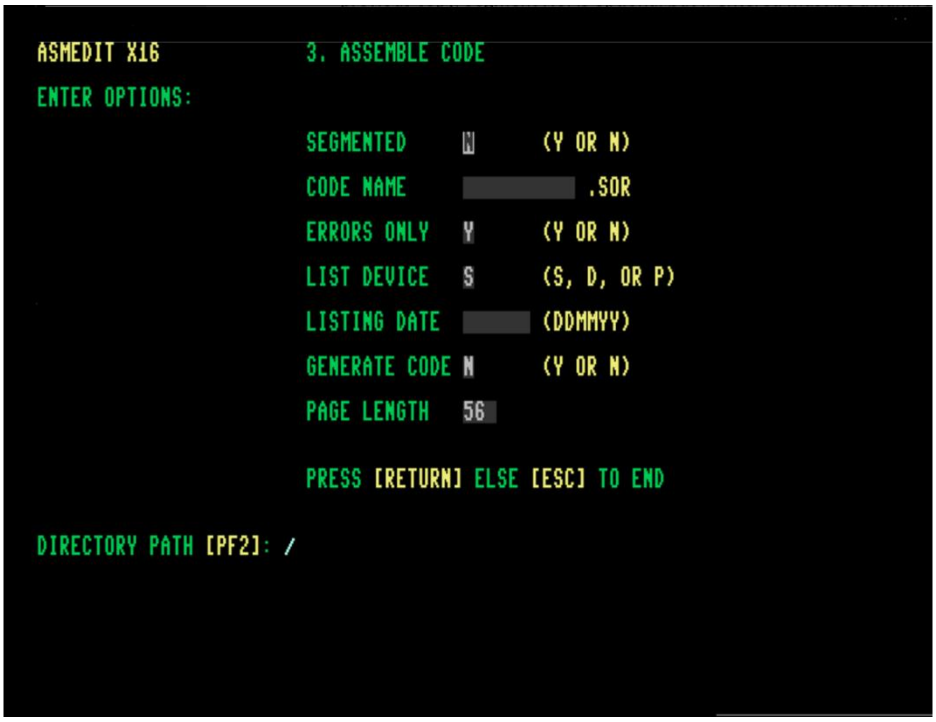
There are two buffers in which source code may be held. The value entered should be 1 or 2 to specify which buffer is to be used.

Press [RETURN] to execute function otherwise [ESC] to return to the PRIMARY COMMANDS menu.

Note: The current DIRECTORY PATH is displayed ('/' means the root). Pressing [F2] allows the user to change the current directory at any point.

3.3 ASSEMBLE CODE (MENU 3)

This menu receives the assembly options and performs the code assembly.



```
ASMEDIT X16              3. ASSEMBLE CODE
ENTER OPTIONS:

                         SEGMENTED     N      (Y OR N)
                         CODE NAME            .SOR
                         ERRORS ONLY   Y      (Y OR N)
                         LIST DEVICE   S      (S, D, OR P)
                         LISTING DATE         (DDMMYY)
                         GENERATE CODE N      (Y OR N)
                         PAGE LENGTH   56

                         PRESS [RETURN] ELSE [ESC] TO END

DIRECTORY PATH [PF2]: /
```

The menu displays the following input areas:

SEGMENTED (Y or N)

Type 'Y' to indicate the source code is held in multiple files prefixed with a segment number, the first file having a SEGMENT NO. of ‘1 ’; or 'N' if held on a single file. If an incorrect value for a program is entered then a FILE NOT FOUND message will likely result.

CODE NAME

The CODE NAME is used to identify the program files on disk.

Source code file names will be formatted as

[SEGMENT NO.][CODE NAME].SOR	(segmented)
[CODE NAME].SOR	(non-segmented).

Executable Code file names will be formatted

[CODE NAME].PRG

Listing files names (when LIST DEVICE is ‘D’, for disk) will be formatted

[CODE NAME].LST

ERRORS ONLY (Y or N)

'Y' will list errors only. 'N' will give a full listing. Ref. *SAMPLE LISTING* for an example of a program listing.

LIST DEVICE (S, D OR P)

'S' (Screen) will display the listing on the screen as it is produced. This is a straight listing with assembly print commands (i.e., skip, eject) suppressed. Use [CTRL] key to slow the screen scrolling.

'D' (Disk) will save the listing to disk as a text file. honouring the print commands specified in the source code.

'P' (Print) will print the listing with page headings, honouring the print commands specified in the source code.

LISTING DATE (DDMMYY)

A date to be placed in the listing page headers.

GENERATE CODE (Y or N)

'Y' instructs the assembler to write executable code to the current directory. This code can be loaded later with the BASIC command LOAD "*code-name*.PRG".

'N' specifies that no code is to be generated, only a listing (full or errors only) is required.

Press [RETURN] to execute function otherwise [ESC] to return to the PRIMARY COMMANDS menu.

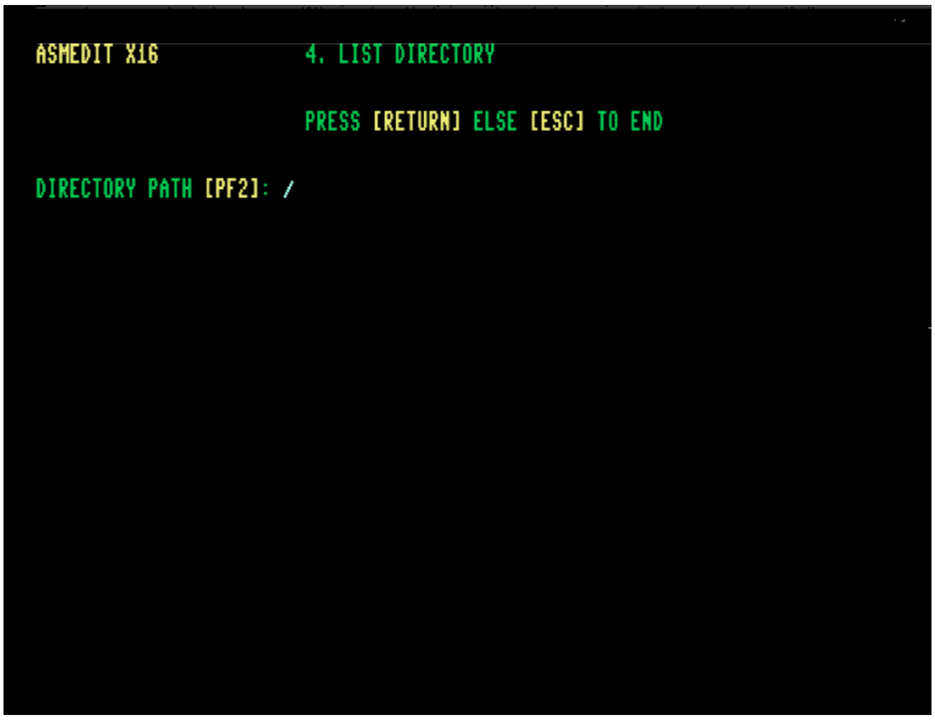
Note: The assembly may be temporarily suspended with [STOP] after which it may be restarted with [RETURN] or cancelled with [ESC]. Pausing the assembly is useful when the list is displayed on the screen rather than being printed.

Note: Refer to 5. THE ASSEMBLER for an overview of this process.

Note: The current DIRECTORY PATH is displayed ('/' means the root). Pressing [F2] allows the user to change the current directory at any point.

3.4 LIST DISK DIRECTORY (MENU 4)

This function simply lists the files held in the current directory as indicated by DIRECTORY PATH.



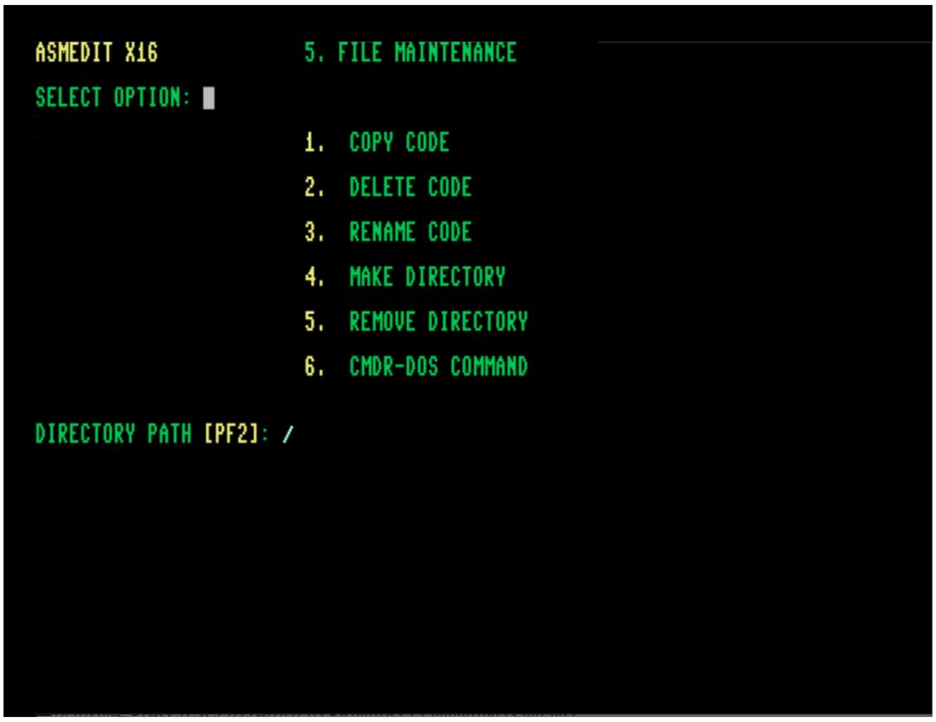
```
ASMEDIT X16             4. LIST DIRECTORY

                        PRESS [RETURN] ELSE [ESC] TO END

DIRECTORY PATH [PF2]: /
```

Press [RETURN] to list the current DIRECTORY otherwise [ESC] to return to the PRIMARY COMMANDS menu.

Note: The current DIRECTORY PATH is displayed on each menu ('/' means the root). Pressing [F2] allows the user to change the current directory at any point.

3.5 FILE MAINTENANCE (MENU 5)

This function displays a list of file maintenance functions which are supported by CMDR-DOS.



```
ASMEDIT X16          5. FILE MAINTENANCE
SELECT OPTION: █

                     1.  COPY CODE
                     2.  DELETE CODE
                     3.  RENAME CODE
                     4.  MAKE DIRECTORY
                     5.  REMOVE DIRECTORY
                     6.  CMDR-DOS COMMAND

DIRECTORY PATH [PF2]: /
```

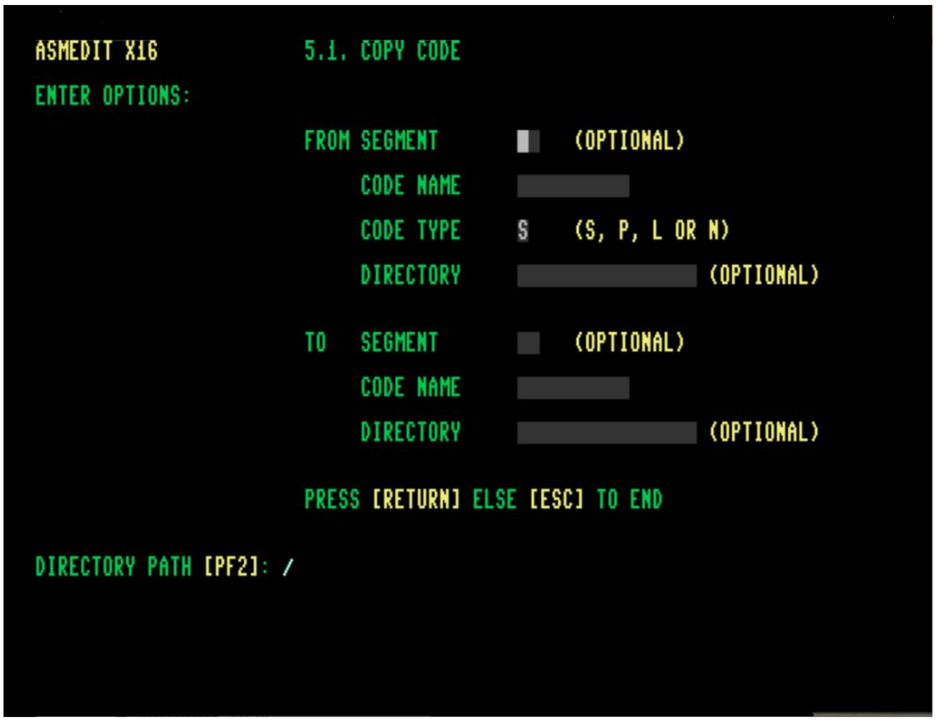
A maintenance function is selected by keying the adjacent number. i.e., Press [2] to a delete code file.

The CODE NAME is used to identify the listing fon disk.

Note: The current DIRECTORY PATH is displayed on each menu ('/' means the root). Pressing [F2] allows the user to change the current directory at any point.

A detailed description of each function now follows from the next page.

3.6 COPY CODE (MENU 5.1)



```
ASMEDIT X16              5.1. COPY CODE
ENTER OPTIONS:

                         FROM SEGMENT          (OPTIONAL)
                              CODE NAME
                              CODE TYPE     S   (S, P, L OR N)
                              DIRECTORY                  (OPTIONAL)

                         TO   SEGMENT          (OPTIONAL)
                              CODE NAME
                              DIRECTORY                  (OPTIONAL)

                         PRESS [RETURN] ELSE [ESC] TO END

DIRECTORY PATH [PF2]: /
```

This function will copy of an ASMEDIT file, or other file if CODE TYPE 'N' is selected. See below.

SEGMENT & CODE NAME (FROM & TO)

These fields are appended together to form a filename prefix, however SEGMENT is optional. If SEGMENT is entered it must be a number from 1 to 99.

CODE TYPE

The type of file to be copied. A File type is appended to the filename prefix as indicated:

'S'	SOURCE type	e.g., '.SOR'
'P'	PROGRAM type	e.g., '.PRG'
'L'	LIST type	e.g., '.LST'
'N'	No file type to be appended	

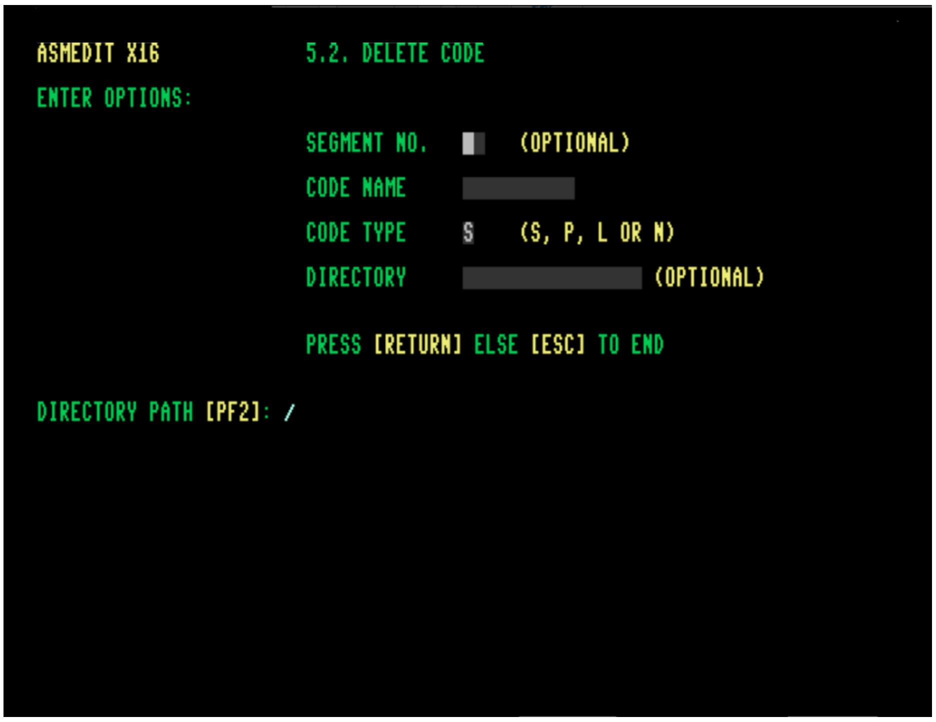
DIRECTORY (FROM & TO)

This field is optional and refers to a child directory below the current DIRECTORY. However double dots '**..**' can be specified to refer to the parent directory. Leaving this field blank means the file is for the current DIRECTORY PATH only.

Press [RETURN] or [ESC] to return to the 5. DISK MAINTENANCE menu.

Note: The current DIRECTORY PATH is displayed on each menu ('/' means the root). Pressing [F2] allows the user to change the current directory at any point.

3.7 DELETE CODE (MENU 5.2)



```
ASMEDIT X16            5.2. DELETE CODE
ENTER OPTIONS:

                       SEGMENT NO.       (OPTIONAL)
                       CODE NAME
                       CODE TYPE     S   (S, P, L OR N)
                       DIRECTORY                   (OPTIONAL)

                       PRESS [RETURN] ELSE [ESC] TO END

DIRECTORY PATH [PF2]: /
```

This function will delete an ASMEDIT file, or other file if CODE TYPE 'N' is selected. See below

SEGMENT NO. & CODE NAME

These fields are appended together to form a filename prefix, however SEGMENT is optional. If SEGMENT is entered it must be a number from 1 to 99.

CODE TYPE

The type of file to be copied. A File type is appended to the filename prefix as indicated:

'S'	SOURCE type	e.g., '.SOR'
'P'	PROGRAM type	e.g., '.PRG'
'L'	LIST type	e.g., '.LST'
'N'	No file type to be appended	

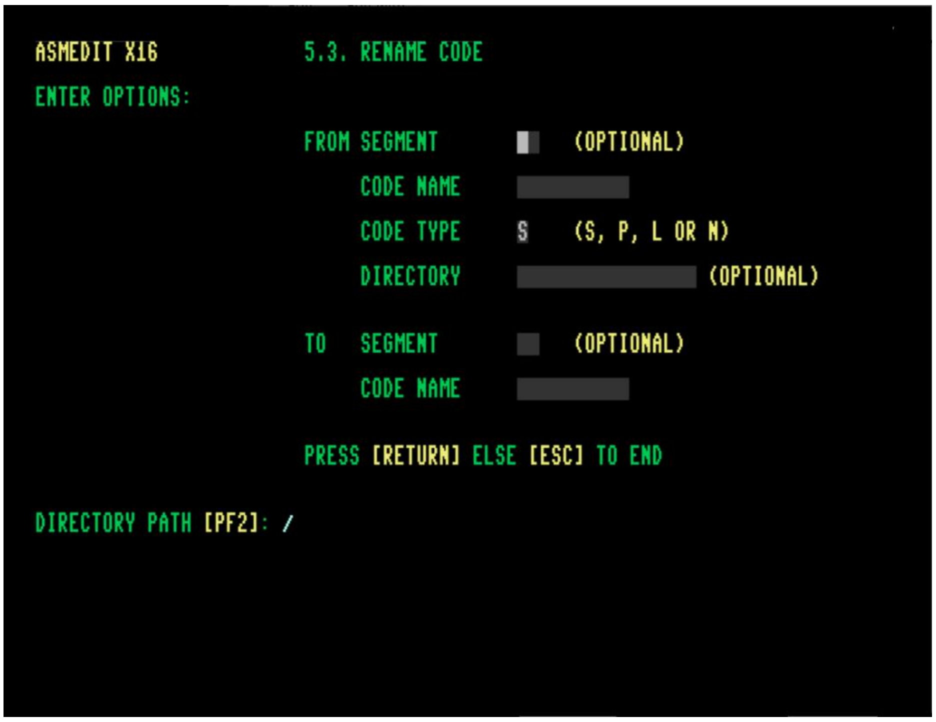
DIRECTORY

This field is optional and refers to a child directory below the current DIRECTORY. However double dots '**..**' can be specified to refer to the parent directory. Leaving this field blank means the file is for the current DIRECTORY PATH only.

Press [RETURN] or [ESC] to return to the 5. DISK MAINTENANCE menu.

Note: The current DIRECTORY PATH is displayed on each menu ('/' means the root). Pressing [F2] allows the user to change the current directory at any point.

3.8 RENAME CODE (MENU 5.3)



```
ASMEDIT X16          5.3. RENAME CODE
ENTER OPTIONS:

                     FROM SEGMENT        (OPTIONAL)
                          CODE NAME
                          CODE TYPE   S    (S, P, L OR N)
                          DIRECTORY              (OPTIONAL)

                     TO   SEGMENT        (OPTIONAL)
                          CODE NAME

                     PRESS [RETURN] ELSE [ESC] TO END

DIRECTORY PATH [PF2]: /
```

This function renames an ASMEDIT file, or other file if CODE TYPE ‘N’ is selected. See below

SEGMENT NO. & CODE NAME

These fields are appended together to form a filename prefix, however SEGMENT is optional. If SEGMENT is entered it must be a number from 1 to 99.

CODE TYPE

The type of file to be copied. A File type is appended to the filename prefix as indicated:

‘S’	SOURCE type	e.g., ‘.SOR’
‘P’	PROGRAM type	e.g., ‘.PRG’
‘L’	LIST type	e.g., ‘.LST’
‘N’	No file type to be appended	

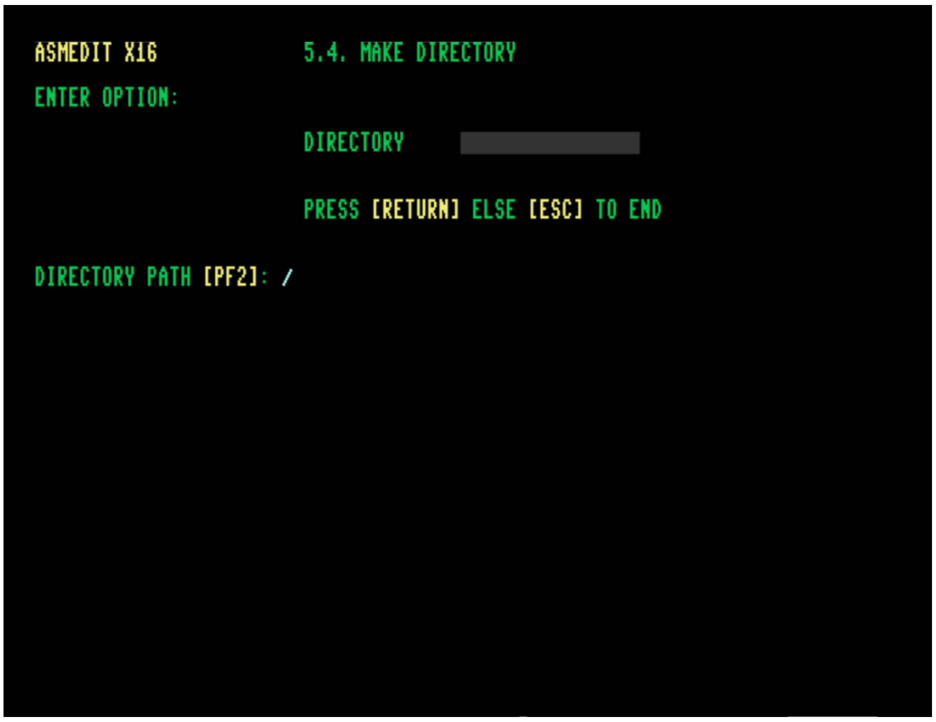
DIRECTORY

This field is optional and refers to a child directory below the current DIRECTORY. However double dots ‘**..**’ can be specified to refer to the parent directory. Leaving this field blank means the file is for the current DIRECTORY PATH only.

Press [RETURN] or [ESC] to return to the 5. DISK MAINTENANCE menu.

Note: The current DIRECTORY PATH is displayed on each menu (‘/’ means the root). Pressing [F2] allows the user to change the current directory at any point.

3.9 MAKE DIRECTORY (MENU 5.4)



```
ASMEDIT X16            5.4. MAKE DIRECTORY
ENTER OPTION:
                       DIRECTORY

                       PRESS [RETURN] ELSE [ESC] TO END

DIRECTORY PATH [PF2]: /
```

This will create a child DIRECTORY below the current DIRECTORY PATH.

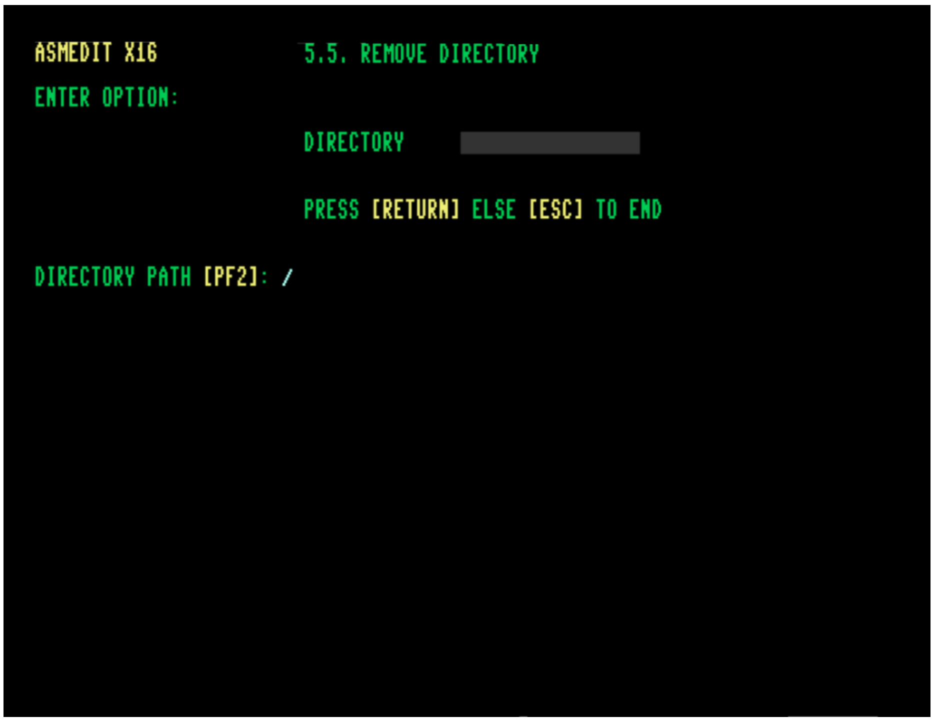
DIRECTORY

This refers to a child directory to be created below the current DIRECTORY.

Press [RETURN] or [ESC] to return to the 5. DISK MAINTENANCE menu.

Note: The current DIRECTORY PATH is displayed on each menu ('/' means the root). Pressing [F2] allows the user to change the current directory at any point.

3.10 REMOVE DIRECTORY (MENU 5.5)


```
ASMEDIT X16          5.5. REMOVE DIRECTORY
ENTER OPTION:
                     DIRECTORY

                     PRESS [RETURN] ELSE [ESC] TO END

DIRECTORY PATH [PF2]: /
```

This will remove a child DIRECTORY below the current DIRECTORY PATH.

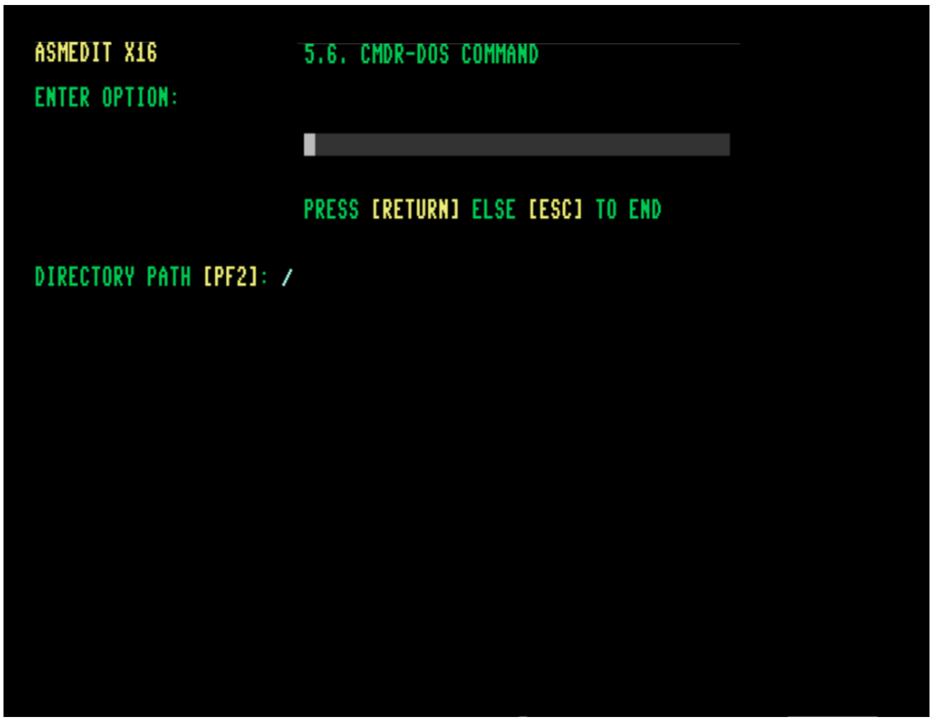
DIRECTORY

This refers to a child directory to be removed below the current DIRECTORY.

Press [RETURN] or [ESC] to return to the 5. DISK MAINTENANCE menu.

Note: The current DIRECTORY PATH is displayed on each menu ('/' means the root). Pressing [F2] allows the user to change the current directory at any point.

3.11 CMDR-DOS COMMAND (MENU 5.6)



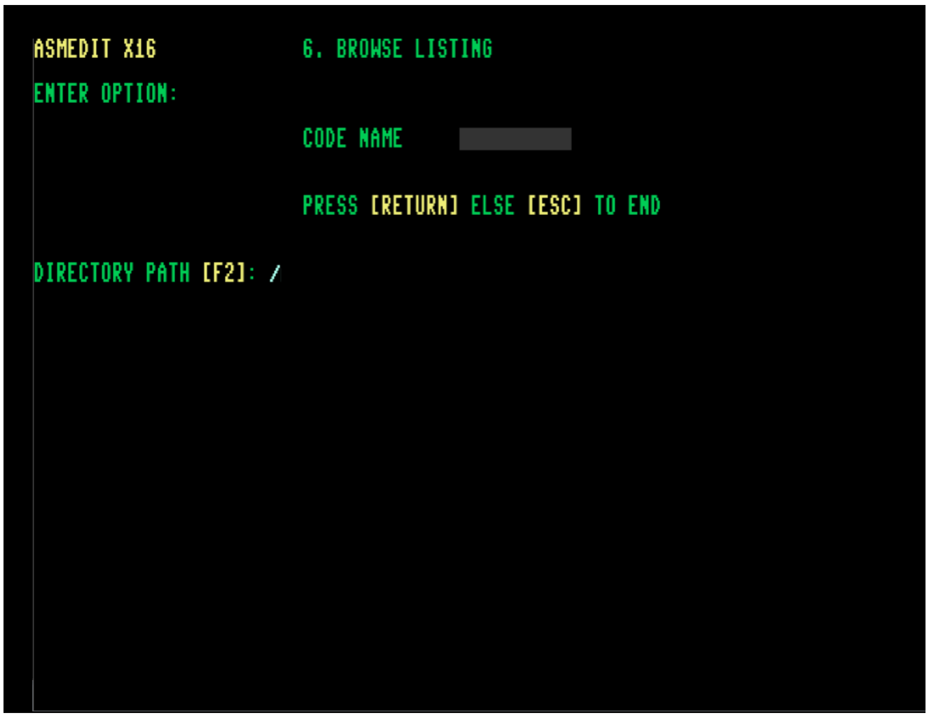
```
ASMEDIT X16              5.6. CMDR-DOS COMMAND
ENTER OPTION:



                         PRESS [RETURN] ELSE [ESC] TO END

DIRECTORY PATH [PF2]: /
```

The input to this function is the same as the DOS command content, which might be coded between double quotes, (with the exception of list directory commands e.g., “$”.

Press [RETURN] or [ESC] to return to the 5. DISK MAINTENANCE menu.

Note: The current DIRECTORY PATH is displayed on each menu (‘/’ means the root). Pressing [F2] allows the user to change the current directory at any point.

3.12 BROWSE LISTING (MENU 6)



```
ASMEDIT X16             6. BROWSE LISTING
ENTER OPTION:

                        CODE NAME

                        PRESS [RETURN] ELSE [ESC] TO END

DIRECTORY PATH [F2]: /
```

This function enables the user to browse a listing held on disk previously created by the Assembler.

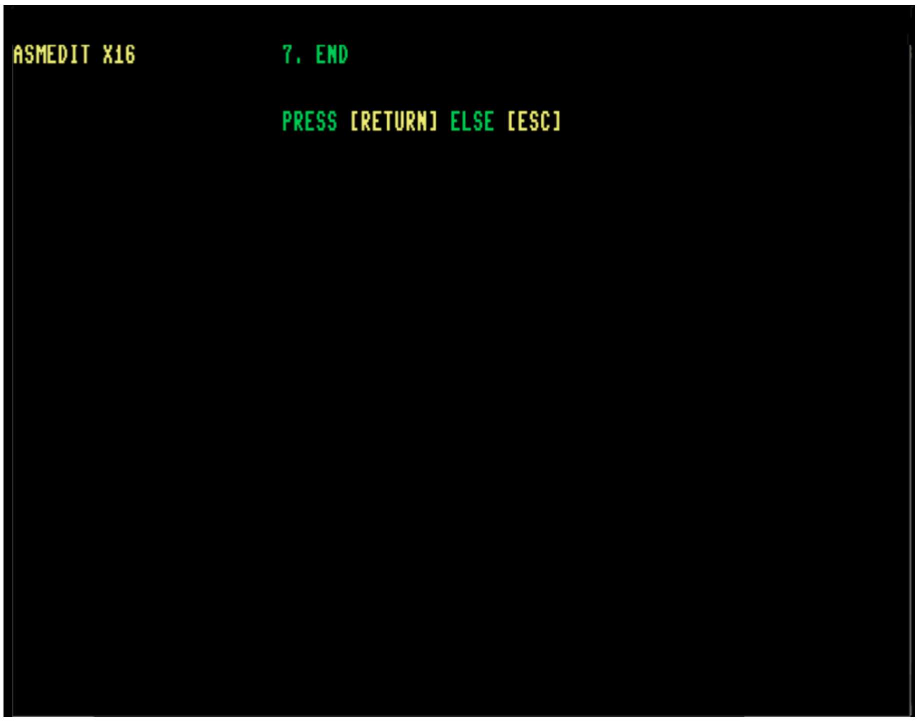
Enter the CODE NAME to identify the listing on disk which will have a file name of ‘*code-name*.LST’.

Press [RETURN] or [ESC] to return to the PRIMARY COMMANDS menu.

Note: The current DIRECTORY PATH is displayed on each menu (‘/’ means the root). Pressing [F2] allows the user to change the current directory at any point.

3.13 END (MENU 7)

This function terminates the program with any code held in the buffers lost.



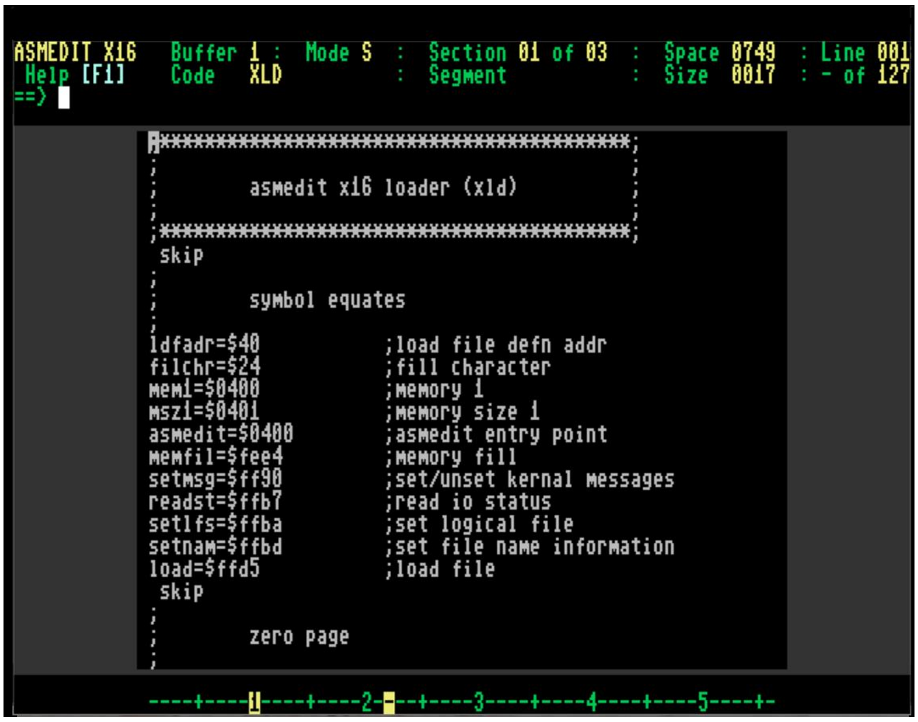
```
ASMEDIT X16              7. END

                         PRESS [RETURN] ELSE [ESC]
```

On selecting '6. END' the option to return to the PRIMARY COMMANDS menu is given (pressing [ESC]); or [RETURN] ends the program ultimately. The program must be reloaded before it can be used again.

Note: Care should be taken before finalising this command to file any code in the buffers that may be required again. To do this return to the LOAD CODE menu, press [RETURN] then exit with [SHIFT]+[X]. If a CODE NAME has not been specified, navigate to the CREATE CODE menu, enter a CODE NAME, ensure that CLEAR BUFFER specifies 'N', press [RETURN] then exit with [SHIFT]+[X]. The code will be secured on disk.

4. THE EDITOR

The EDITOR is entered from the CREATE CODE or LOAD CODE menu. This function allows the user to enter, display and manipulate source code (ref.5.2 SOURCE CODE FORMAT). During this function source code is held in memory otherwise known as a 'buffer'. Two buffers are used (Ref. BUFFER in the appendices).



```
ASMEDIT X16   Buffer 1 :  Mode S  :  Section 01 of 03  :  Space 0749  : Line 001
 Help [F1]    Code   XLD          :  Segment           :  Size  0017  : - of 127
==>

            ;****************************************;
            ;                                        ;
            ;        asmedit x16 loader (xld)        ;
            ;                                        ;
            ;****************************************;
             skip
            ;
            ;        symbol equates
            ;
            ldfadr=$40            ;load file defn addr
            filchr=$24            ;fill character
            mem1=$0400            ;memory 1
            msz1=$0401            ;memory size 1
            asmedit=$0400         ;asmedit entry point
            memfil=$fee4          ;memory fill
            setmsg=$ff90          ;set/unset kernal messages
            readst=$ffb7          ;read io status
            setlfs=$ffba          ;set logical file
            setnam=$ffbd          ;set file name information
            load=$ffd5            ;load file
             skip
            ;
            ;        zero page
            ;

            ----+----1----+----2----+----3----+----4----+----5----+-
```

Usually, a program is defined with one SEGMENT having up to 99 SECTIONs, each SECTION having a maximum of 127 lines. However, large programs may be defined as linked segments held on separate disk files. The EDITOR can handle only one SEGMENT (per BUFFER) at a time however all linked segments will be assembled into one execution file.

Note: Pressing [F1] at any point in the EDITOR gives a QUICK REFERENCE to the commands. Press [ESC] to return to the EDITOR

The following values are displayed in the header lines:

BUFFER	1 or 2, depending on the current buffer selection
CODE	The name of the program to be coded.
SEGMENT	Either blank (single segment) of nn (01-99) representing the segment identifier.
MODE	Displays which mode is in operation. i.e., S: SCAN EDIT, O: OVERWRITE or I: INSERT. Scan mode allows for commands to be entered, whereas the two input modes can be to OVERWITE or INSERT.
*SPACE and SIZ*E	Displayed in Blocks (256 bytes). The two buffers combined have a maximum of 192K (768 blocks) available.
==>	The Command Prompt.

**** NOTE: A QUICK REFERENCE GUIDE to commands is given in EDITOR COMMANDS QUICK REFERENCE in the Appendices.*

4.1 COMMAND SYNTAX SYMBOLS

[ ]	Keystroke
[ ] + [ ]	2 keys at once
‘ ‘	Enter character string
{ }	Repeatable action(s)
/	OR
( )	Group of actions

4.2 KEYSTROKE COMMANDS

The Editor has the following single stroke commands:

Command Quick Reference	[F1]	
Step forward one section	[PGDN]	mode S only
Step backward one section	[PGUP]	mode S only
Cursor left	[CURSOR LEFT]	
Cursor left (rapid)	[CURSOR LEFT] + [SHIFT]	
Cursor right	[CURSOR RIGHT]	
Cursor right (rapid)	[CURSOR RIGHT] + [SHIFT]	
Cursor up	[CURSOR UP]	
Cursor up (rapid)	[CURSOR UP] + [SHIFT]	
Cursor down	[CURSOR DOWN]	
Cursor down (rapid)	[CURSOR DOWN] + [SHIFT]	
Set Tab or Delete Tab	[TAB] + [CTRL]	
Skip to next tab position	[TAB]	
Skip to previous tab position	[TAB] + [SHIFT]	
Select Text Display Option	[F12]	

Note: Some of these commands repeat if the key is held down.

4.3 FIND

[F] { 'String' [RETURN] { [RETURN] } } [ESC]

This command searches for a given string of characters. The search is carried out left to right and downward from the current input cursor position. On reaching the end of code the search begins again from the first line. If a matching string is not found, a Condition Code of 2 is returned otherwise the key input is positioned at the start of the matching code. Pressing [RETURN] again will locate the next occurrence. The function is ended with [ESC]. The maximum length that can be entered is 30 characters.

Example, to find the occurrence of 'bananas', Press [F], enter **bananas** *then [RETURN] (Use [BACKSPACE] to correct errors). The input cursor will point to the first character of the string entered - the 'b' in bananas. Pressing [RETURN] again will point to the next occurrence, if found.*

Note: Press [STOP] to terminate the command when in operation

4.4 ALTER

[A] 'old string' [RETURN] 'new string' [RETURN] ([A] [RETURN]) / ([F] { [RETURN] }) [ESC]

This enables a selected string of characters to be replaced with another. The replacement can either be selective or can automatically replace all occurrences. After pressing [A] and entering the ‘old-string’ then [RETURN], the ‘new-string’ and [RETURN] again, there are two alternatives. First, by pressing [A] all occurrences of the string will be altered. Or by pressing [F] each occurrence is found and may be changed by pressing [RETURN], or 'F' again to move to the next occurrence. The function is cleared with the [ESC] key.

Example, throughout the code held in the Edit buffer, we wish to change all occurrences of the word ‘program’ to ‘programme’: Press [A] enter **program** *[RETURN] screen shows* **A program***. Enter programme [RETURN] (Use [BACKSPACE] to correct errors) screen shows* **A program;programme***. Now pressing [A] will change all occurrences of 'program'. Or, alternatively, pressing 'F' will locate an occurrence, which can the either be altered by pressing [RETURN] or can be skipped by pressing [F] again. It will be found that all occurrences of any 'string' can quickly be skipped through by pressing [F]. NOTE: A problem can occur when altering, in that changing a word such as 'the' will also change part of 'then'. The 'fix' is to space where necessary. For example, if one altered 'the ' (space following e) then words containing 'the' such as 'then' will not be affected.*

Note: Press [STOP] to terminate the command when in operation

4.5 DELETE LINES

[D] start [RETURN] end [RETURN]

This command deletes consecutive lines between two cursor positions. Cursor position is achieved using the cursor control keys. To delete lines press [D] then position the cursor onto the top line of the code to be deleted and press [RETURN]. Then, again using the cursor control keys, position the cursor so that it is on the bottom line to be deleted, and press [RETURN]. *Note: A single line may be deleted by positioning the cursor on the same line and pressing [RETURN] twice*. The whole block is deleted. An error Condition Code of 3 is returned if the end line is before the start line, in which case press [ESC] and try again.

******* *NOTE: You do not enter the line numbers manually - positioning the cursor and pressing [RETURN] automatically selects the line.*

4.6 COPY LINES

[C] start [RETURN] end [RETURN] { insert-after [RETURN] } [ESC]

This command copies consecutive lines between two cursor positions to another location or locations. Cursor position is achieved using the cursor control keys. To copy lines press [C] then position the cursor onto the top line of the code to be copied and press [RETURN]. Then, again using the cursor control keys, position the cursor so that it is on the bottom line to be copied, and press [RETURN]. *Note: A single line may be copied by positioning cursor on the same line and pressing [RETURN] twice*. Using the cursor and page control keys, select the line after which the ‘copied’ lines are to be inserted and press [RETURN] to perform the copy. This last action may be repeated to copy lines to different locations in potentially different sections. An error Condition Code of 3 is returned if the end line is before the start line, in which case press [ESC] and try again. The [ESC] terminates the function.

**** NOTE: You do not enter the line numbers manually - positioning the cursor and pressing [RETURN] automatically selects the line.*

4.7 MOVE LINES

[M] start [RETURN] end [RETURN] insert-after [RETURN]

This command moves consecutive lines between two cursor positions to another location. Cursor position is achieved using the cursor control keys. To copy a block of code press [M] then position the cursor onto the top line of the code to be moved and press [RETURN]. Then, again using the cursor control keys, position the cursor so that it is on the bottom line to be moved, and press [RETURN]. *Note: A single line may be moved by positioning the cursor on the same line and pressing [RETURN] twice*. Using the cursor and page control keys, select the line after which the ‘moved’ lines are to be inserted and press [RETURN} to perform the move. An error Condition Code of 3 is returned if the end line is before the start line, in which case press [ESC] and try again.

**** NOTE: You do not enter the line numbers manually - positioning the cursor and pressing [RETURN] automatically selects the line.*

4.8 TRANSCRIBE LINES

[T] start [RETURN] end [RETURN] AUTOSWAP { insert-after [RETURN] } [ESC]

This command is initiated with [T]. The operation of the command is identical to the COPY command, except that it ‘transcribes’ lines from the EDIT buffer to a location or locations in the STORAGE buffer. The 'start' and 'end' lines are entered as with the copy, then on pressing [RETURN] the buffers automatically 'SWAP' over so that the selected lines may be inserted in one or more segments in the other buffer. The [ESC] terminates the function.

**** NOTE: You do not enter the line numbers manually - positioning the cursor and pressing [RETURN] automatically selects the line.*

4.9 SWAP BUFFERS

[S]

This simply exchanges the total contents of the EDIT buffer with the contents of the STORAGE buffer. The only way to put code into the STORAGE buffer is by 'SWAPping' the EDIT buffer contents.

4.10 COPY- A COLUMN

[SHIFT]+[C] start [RETURN] end [RETURN] { place-at [RETURN] } [ESC]

This command copies a column of code/text between two cursor positions, as illustrated below, to another location overwriting the existing content. Cursor position is achieved using the cursor and page control keys. To copy a column press [SHIFT]+[C] then move the cursor to the top left corner representing the ‘start’ position and press [RETURN]. Similarly move the cursor to the bottom right corner which is the ‘end’ position. The 'place at' is the new start for the whole column specified using the page and control keys as required. Columns are repeatable by just moving the cursor to another start position and pressing [RETURN] again. There is no limitation on the new starting position for the column and so code/text can be overlaid as desired. The [ESC] terminates the function.

```
        +..........          +..........
       /.         .        / .         / OR place at
      / . XXXXXXX .  place/  . XXXXXXX/.
start/  . XXXXXXX .     at   . XXXXXX/ .
        . XXXXXXX .          . XXXX+/.........
        . XXXXXXX .          . XXXX.         .
        .         .          .     . XXXXXXX .
        ..........+          ....... XXXXXXX .
                 /                 . XXXXXXX .
                /                  . XXXXXXX .
            end/                   .         .
                                   ...........
```

4.11 MOVE- A COLUMN

[SHIFT]+[M] start [RETURN] end [RETURN] place-at [RETURN]

This command moves a column of code/text between two cursor positions to another location overwriting the existing content. Cursor position is achieved using the cursor and page control keys. To move a column press [SHIFT]+[M] then move the cursor to the top left corner representing the ‘start’ position and press [RETURN]. Similarly move the cursor to the bottom right corner which is the ‘end’ position. The 'place at' is the new start for the whole column specified using the page and control keys as required. The original location of the column is replaced by blanks.

4.12 TRANSCRIBE- A COLUMN

[SHIFT]+[T] start [RETURN] end [RETURN] AUTOSWAP { place-at [RETURN] } [ESC]

This command is initiated with [SHIFT]+[T]. The operation of the command is identical to the COPY COLUMN command, except that it ‘transcribes’ a column from the EDIT buffer to a specified location or locations in the STORAGE buffer. The 'start' and 'end' positions are entered as with the copy, then on pressing [RETURN] the buffers automatically 'SWAP' over so that the column may be placed in the other buffer as with the copy command. The [ESC] terminates the function.

4.13 COPY- A SECTION

[ALT]+[C] from [RETURN] { insert-after [RETURN] } [ESC]

The page control keys, [PGUP] and [PGDN], which enable movement between sections, or the GO command may be used to select the section to be copied. The position of the Input Cursor is not significant. The current section number is always displayed at the top of the screen. When the required section is on view [RETURN] should be pressed. Similarly, the section AFTER which the copy is to be inserted may be selected in the same way. On pressing [RETURN] the 'copied' section will be inserted. Subsequent [RETURN]s will produce further copies always after the current section on display. The [ESC] terminates the function.

4.14 MOVE- A SECTION

[ALT]+[M] from [RETURN] insert-after [RETURN]

The page control keys, [PGUP] and [PGDN], which enable movement between sections, or the GO command may be used to select the section to be moved. The position of the Input Cursor is not significant. The current section number is always displayed at the top of the screen. When the required section is on view [RETURN] should be pressed. Similarly, the section AFTER which the 'moved' section is to be inserted may be selected in the same way. On pressing [RETURN] the 'moved' section will be inserted

4.15 TRANSCRIBE- A SECTION

[ALT]+[T] from [RETURN] AUTOSWAP { insert-after [RETURN] } [ESC]

The operation is the same as for the copy except that an automatic 'SWAPping' of buffers occurs after registering the source section. This then gives the facility to transcribe a section of code from one program or segment, say in *BUFFER* 1, into another program or segment in *BUFFER* 2. The [ESC] terminates the function.

4.16 INSERT- A SECTION

[ALT]+[I] [RETURN]

Inserts section AFTER current section on view.

4.17 DELETE- A SECTION

[ALT]+[D] [RETURN]

Deletes current section on view unless there is only one section. ASMEDIT does not allow a buffer to contain zero sections but the remaining section content may be cleared with the ERASE command.

4.18 ERASE- A SECTION

[SHIFT]+[E] [RETURN]

Clears current section on view.

4.19 GO TO A SECTION

[G] section number [RETURN]

Section specified becomes current section displayed.

4.20 UNDO- CURRENT SECTION

[U] [RETURN]

The current section on view is held in the Edit buffer. If a significant formatting error occurs on this page, this command may be used to refresh the section from the Main buffer pool to effectively undo all changes made since the section was first displayed. It can also be used to reverse the ERASE command.

4.21 VERIFY CODE

[V] 0 / 1 [RETURN]

The command performs an assembly of the source code in the current *BUFFER* for verification purposes. [ESC] to cancel command before [RETURN]. The options are: -

Option 0 (or no option)	- display errors only
Option 1	- displays the full listing on screen

NOTE: This command does not allow for segmented code to be verified. Segmented code is held across multiple files and therefore is incomplete within the BUFFER area. Primary Command Menu 3 should be used instead.

4.22 EXIT & OPTIONALLY SAVE CODE TO DISK

[X] or [SHIFT]+[X]

Pressing [X] returns control to Primary Commands. If [SHIFT]+[X] is pressed (and the CODE NAME is defined) the code will also be written to disk as a '.SOR' file.

4.22 PASS CONTROL TO INPUT COMMANDS

[F3]

Pressing [F3] passes control to the input mode so that code can be typed into the buffer from the keyboard.

4.24 INPUT COMMANDS

OVERVIEW

Apart from typing into the Edit buffer, a number of 'key stroke' commands are available. These include setting of tabs, skipping between tabs, character deletion, inserting blank lines and toggling the input mode between 'Insert' and 'Overwrite'. Four-way scanning is available using the cursor control keys. For rapid action press shift key as well.

SELECT SUB-MODE

[INST]

On entry, the editor will be in the 'O' input mode, which is OVERWRITE. Typing over code/text in this mode will replace the characters with whatever is entered from the keyboard. Pressing [INST] will change the input mode to 'I', for INSERT. In this mode space will be made for characters being typed, and all code/text to the right of the cursor will be moved along. Pressing [INST] again will change the mode back to OVERWRITE.

Note: In INSERT mode, If the last non-blank character on the current line is at the last column position, indicating that the line is full, then input is supressed for this line.

SETTING TABS

[CTRL]+[TAB]

Tab positions are pre-selected for formatting code. These positions may be changed, removed or added to by this command. Moving the cursor and pressing [CTRL]+[TAB] sets a tab, except where a tab position is already set, in which case it is removed.

When editing: -

[RETURN] moves the cursor to the FIRST tab position on the next line.

[SHIFT]+[RETURN] moves the cursor to column 1 of the next line.

[TAB] advances the cursor to the next tab position. If the cursor is positioned at the last tab the cursor is advanced to the first tab position of the next line.

[SHIFT]+[TAB] will move the cursor to the previous tab. If the cursor is positioned at the first tab the cursor is moved up a line to the last tab position.

DELETE CHARACTER

[BACKSPACE] / [DELETE]

[BACKSPACE] will delete characters to the left of the current cursor.

[DELETE] key will perform a 'forward' delete, deleting characters to the right of the cursor.

INSERT A NEW LINE (^)

[SHIFT] + [6] will insert a blank line after the current line. The key repeats if held down.

5. THE ASSEMBLER

5.1 DESCRIPTION

The function of the Assembler is to generate executable code from source code specifically to run on the commander X16. It can also produce a program listing (complete or errors only). Refer to *SAMPLE LISTING* for an example of a program listing. The Assembler will read through the source code TWICE. The first pass identifies SYMBOLs and builds a SYMBOL table. At this point some SYMBOL values will be unresolved. The second pass resolves these values, creates the listing, identifies and reports code errors and generates the required machine code. Finally, the SYMBOLs are reported in tabular form, in alphabetic order at the end of the program listing. The progress of the assembly is displayed on the screen.

Note: The Assembly can be suspended at any time with the [STOP] key and either terminated with the [ESC] key or resumed with the [RETURN] key.

5.2 SOURCE CODE FORMAT

The ASSEMBLER scans each line of code expecting the following format.

((SYMBOL/LABEL)　　MNEMONIC/COMMAND (OPERAND))　　(COMMENTS)

A SYMBOL or LABEL is optional. If entered it must begin in column 1.

All characters on a line following a semicolon (**;**) are treated as COMMENTs

An assembly COMMAND or MNEMONIC code must be preceded by at least one blank.

Depending on the MNEMONIC or COMMAND, an OPERAND preceded by a blank is further required.

For examples of source code lines refer to the program code in *APPENDIX B*.

5.3 SYMBOL & LABEL

A SYMBOL is used to assign a value to a name/identifier. A SYMBOL is recognised by a character string beginning in column 1 which is no more than 8 characters in length and begins with an alphabetic character. Symbols are defined in the format 'symbol=value' however a SYMBOL with no operand in column 1 is known as a LABEL and takes the current program address as its value. Some examples of a SYMBOL definition are ('PRINT=$FFD2', 'EOF=$80', 'START', 'DATA=*'). Once defined a SYMBOL can be used to substitute a particular value in an '65C02 assembler instruction operand', e.g. ('JMP START', 'CMP #EOF', 'JSR PRINT').

5.4 SYMBOL OPERANDS

In addition to a straight numeric value, a SYMBOL value may be derived from one or more of the following:

symbol	Uses the value represented by the SYMBOL.
*	Current program address.
>	Prefixed to a SYMBOL to signify that the high order byte value should be used. (e.g. '>PRINT')
<	Prefixed to a SYMBOL to signify that to the low byte order byte should be used. (e.g., '<START')
$	Prefixed to indicate the subsequent value is in hexadecimal format (e.g., '$d', '$FFE4').
'	Prefix to indicate the value is a character or character string (e.g., #'x').
+value	Extend an operand by adding a further 'value' e.g. ('START+2').
-value	Extend an operand by subtracting a further 'value' e.g. ('*-2').

5.5 GENERAL ASSEMBLY COMMANDS

Must be coded AFTER column 1 with only one command on a line.

SKIP n	Applies to listing only. Skip n lines where n=1 to 3 or blank, defaulting to 1
EJECT	Applies to listing only. Skip to start of next page
END	Signifies the end of the source code to be assembled.
*=value	This command sets the program counter (address). The command is required first to signify the start of ZERO PAGE. ZERO PAGE may then be changed or incremented with a subsequent command, for example: -

```
                *=0
        DBANK   *=*+1
        EBANK   *=*+1
                *=$22
        SCRADD  *=*+2
```

A further asterisk command is required to signify the start of code. This must be a value **above** the ZERO PAGE (>$FF) and be specified **once only**.

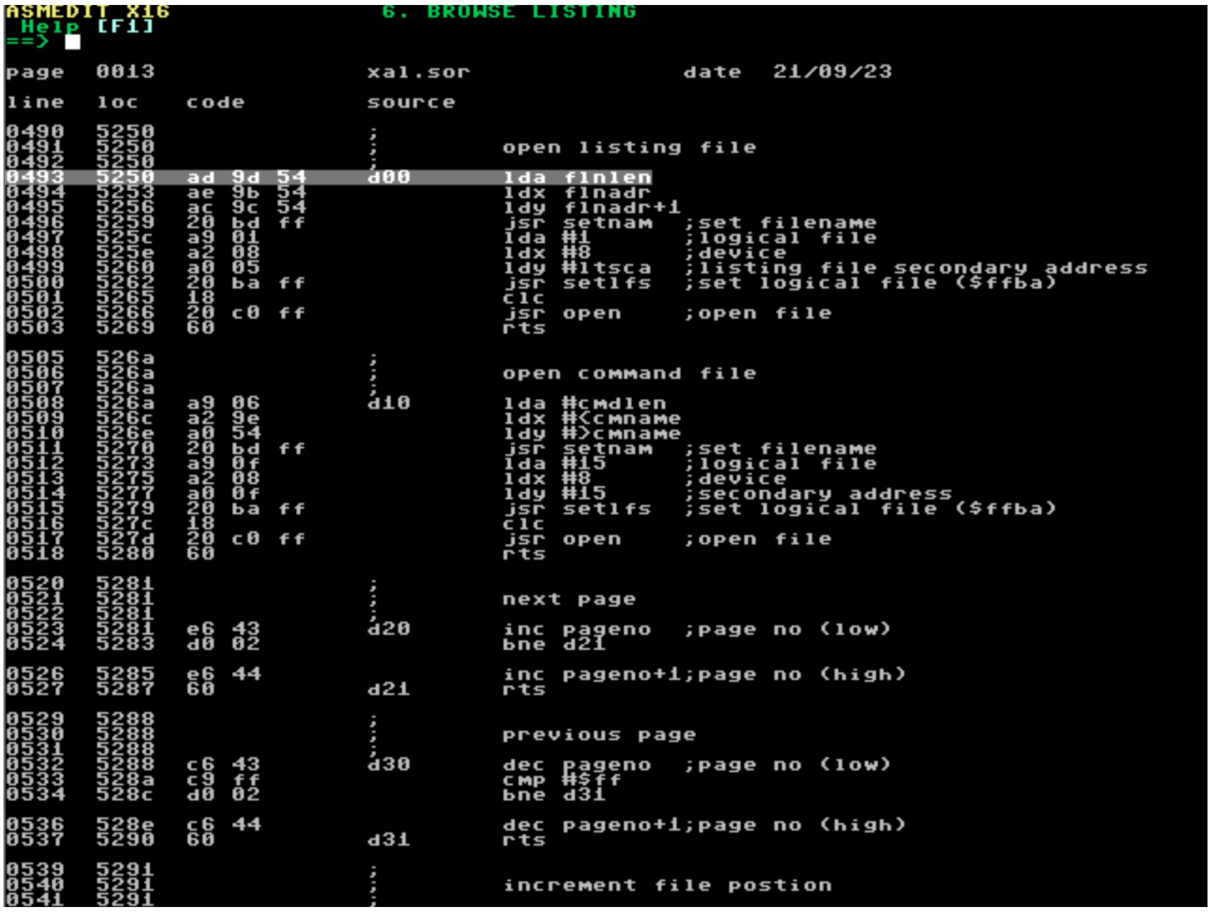
```
                *=$800
```

5.6 SPECIAL ASSEMBLY COMMAND

+	A blank line with only a single '+' IN column 1 instructs the assembler to load the next SEGMENT from disk (e.g., '2 SORT.SOR' is loaded if the current SEGMENT is '1 SORT'.

6. THE BROWSER

6.1 DESCRIPTION


```
ASMEDIT X16                 6. BROWSE LISTING
 Help [F1]
==> █

page  0013              xal.sor              date  21/09/23

line  loc   code        source

0490  5250              ;
0491  5250              ;        open listing file
0492  5250              ;
0493  5250  ad 9d 54    d00      lda flnlen
0494  5253  ae 9b 54             ldx flnadr
0495  5256  ac 9c 54             ldy flnadr+1
0496  5259  20 bd ff             jsr setnam  ;set filename
0497  525c  a9 01                lda #1      ;logical file
0498  525e  a2 08                ldx #8      ;device
0499  5260  a0 05                ldy #ltsca  ;listing file secondary address
0500  5262  20 ba ff             jsr setlfs  ;set logical file ($ffba)
0501  5265  18                   clc
0502  5266  20 c0 ff             jsr open    ;open file
0503  5269  60                   rts

0505  526a              ;
0506  526a              ;        open command file
0507  526a              ;
0508  526a  a9 06       d10      lda #cmdlen
0509  526c  a2 9e                ldx #<cmname
0510  526e  a0 54                ldy #>cmname
0511  5270  20 bd ff             jsr setnam  ;set filename
0512  5273  a9 0f                lda #15     ;logical file
0513  5275  a2 08                ldx #8      ;device
0514  5277  a0 0f                ldy #15     ;secondary address
0515  5279  20 ba ff             jsr setlfs  ;set logical file ($ffba)
0516  527c  18                   clc
0517  527d  20 c0 ff             jsr open    ;open file
0518  5280  60                   rts

0520  5281              ;
0521  5281              ;        next page
0522  5281              ;
0523  5281  e6 43       d20      inc pageno  ;page no (low)
0524  5283  d0 02                bne d21

0526  5285  e6 44                inc pageno+1;page no (high)
0527  5287  60          d21      rts

0529  5288              ;
0530  5288              ;        previous page
0531  5288              ;
0532  5288  c6 43       d30      dec pageno  ;page no (low)
0533  528a  c9 ff                cmp #$ff
0534  528c  d0 02                bne d31

0536  528e  c6 44                dec pageno+1;page no (high)
0537  5290  60          d31      rts

0539  5291              ;
0540  5291              ;        increment file postion
0541  5291              ;
```

The BROWSER is entered from Menu 6. BROWSE LISTING and has the following commands:

6.2 GO TO A PAGE

This command moves position directly to the given page number.

[G] page number [RETURN]

6.3 KEYSTROKE COMMANDS

The Browser has the following single stroke commands:

Command Quick Reference	[F1]
Step forward one section	[PGDN]
Step backward one section	[PGUP]
Cursor up	[CURSOR UP]
Cursor down	[CURSOR DOWN]

Note: Some of these commands repeat if the key is held down.

APPENDIX A

HANDLING CODE

BUFFER

The EDITOR holds source code in so-called BUFFERs. Two buffers are used, buffer 1 and buffer 2. The buffer in which source code is currently in view is referred to in this manual as the EDIT buffer. This may be either buffer 1 or 2. Code entered from the keyboard is always entered into the EDIT buffer, similarly all filing is performed from the EDIT buffer. The code not in the EDIT buffer, held temporarily to one side as it were, is kept in the STORAGE buffer. When in the STORAGE buffer it cannot be saved or 'handled' without first passing it back to the EDIT buffer. However, its whole contents can be swapped with the EDIT buffer at the touch of a key (ref. 4.9 SWAP BUFFERS).

It may be useful to think of buffers 1 and 2 as being a turn table divided into two. The half nearest and currently in use is then the EDIT buffer. The part furthest away is the STORAGE buffer. Swapping the buffers is then equated with turning the table so that the other half is then nearer and so becomes the EDIT buffer.

CODE NAME

The EDITOR uses a file naming convention to aid the saving and loading of source code on disk. Programs are assigned a CODE NAME by the user (this may be done before or after creating the source code but before it’s saved). For single SEGMENT programs the CODE NAME is combined with the character suffix ‘.SOR’ to create a filename: CODE NAME+.SOR. e.g., ‘SORT.SOR’

SEGMENT

Large programs may be broken down into SEGMENTs. Each SEGMENT is then stored as a single file prefixed by a 2-character assigned SEGMENT No. and suffixed by the CODE NAME + ‘. SOR'. The first segment must be 1 and each subsequent SEGMENT must be the previous SEGMENT + 1.

For example, a ‘3 SEGMENT’ program, with an assigned CODE NAME of ‘SORT’ would have the following files: ‘1 SORT.SOR’, ‘2 SORT.SOR’ & ‘3 SORT.SOR’

SECTION

A SEGMENT is divided into SECTIONs. A section having a maximum of 127 Lines. A SEGMENT has a maximum of 99 SECTIONs. All programs must have at least one SECTION.

Note: Source code may be transferred freely between SECTIONs within a SEGMENT. Code may be copied (using one of the TRANSCRIBE commands e.g.,4.8 TRANSCRIBE LINES from one SEGMENT to another. To do this requires both SEGMENTs be loaded. This is achieved by loading one SEGEMNT into the Edit buffer, swapping it into the Storage buffer (ref.4.9 SWAP BUFFERS) and then loading the second SEGMENT into the Edit buffer.

APPENDIX B

QUICK START INTRODUCTION SCRIPT

This is a simple example of ENTERING, EDITING and ASSEMBLING code

ACTION	**COMMENT**
DOS"CD:ASMEDIT"	Makes ASMEDIT the current directory
LOAD "ASMEDIT.PRG" then RUN	ASMEDIT loads and runs entering the PRIMARY COMMANDS menu.
Press '1'	ASMEDIT enters menu 1. CREATE CODE

The cursor is first positioned on the CLEAR BUFFER parameter which can be left as 'Y'.
Use the down cursor key to position on the CODE NAME input area, ignoring SEGMENT NO. for this exercise.

Enter CODE NAME 'HELLOWORLD'	This Identifies the code to be input.
Press [RETURN]	ASMEDIT enters the EDITOR in mode S, 'Scan/Edit'
Press [F3]	ASMEDIT enters into mode O 'Overwrite' to enable input from the keyboard.

Type in perhaps HELLOWORLD as per the SAMPLE CODE below. Ref.to 4.2 KEYSTROKE COMMANDS & 4.24 INPUT COMMANDS to aid with the input and ref. 5. THE ASSEMBLER for an understanding of the syntax. Toggle [INST] to switch between OVERWRITE and INSERT modes as required.

Note the current MODE is displayed in the screen header.

Note: Pressing [F1] at any point in the EDITOR gives a QUICK REFERENCE to the commands. Press [ESC] to return to the EDITOR

When typing is complete then:

Press [ESC]	Exit from Keyboard Input mode back to Scan/Edit mode [S].
Press 'V' followed by [RETURN]	This will perform a VERIFY operation in the form of an assembly to show any errors found.
Press [ESC]	This ends the VERIFY and returns to the EDITOR

Go back to input mode with [F3] to correct any errors. Then from within Scan/Edit mode [S]: -

Press Shifted 'X'	This saves file 'HELLOWORLD.SOR' to disk and exits the EDITOR. If the save is successful ASMEDIT passes control to the LOAD CODE menu otherwise it returns to the CREATE CODE menu.

Press [ESC]	ASMEDIT returns to the PRIMARY COMMANDS menu
Press ‘3’	ASMEDIT enters menu 3. ASSEMBLE CODE.

You do not to need to change the default values unless you want to generate code and/or the listing. For an understanding of the input parameters ref. to 3.3 ASSEMBLE CODE (MENU 3)

ACTION	**COMMENT**
Press [RETURN]	The assembly will begin

If the listing is directed to the screen, then at the end of the assembly The ASSEMBLER gives the option to list the SYMBOLS with [RETURN] or to end with [ESC]

Press [RETURN]	the SYMBOLs are listed in tabular form.
Press [ESC]	ASMEDIT returns to 3. ASSEMBLE CODE menu
Press [ESC]	ASMEDIT return to the PRIMARY COMMANDS menu.
Press ‘6’	ASMEDIT enters the 6. END menu.
Press [RETURN]	ASMEDIT returns control to BASIC.

SAMPLE SOURCE CODE

The sample code is in to parts.

The first sample code is a simple program to reside at address $0400. It can be invoked with LOAD"HELLOWORLD.PRG",8,1 then SYS $0400.

The second part is a 'loader' program which will load and invoke the 'hello world' program.

Sample Part 1 – Hello World (HELLOWORLD.SOR)

```
;
;        hello world sub-routine
;
print=$ffd2
         *=$0400
         ldx #0
loop     lda hello,x
         beq exit
 skip
         jsr print
         inx
         bne loop
 skip
exit     rts
 skip
hello    byt '!!! HELLO WORLD !!!'
         byt 0
 skip
         end
```

Sample Part 2 – The program loader (LOADER.SOR)

```
;*******************************************;
;                                           ;
;        program loader                     ;
;                                           ;
;*******************************************;
 skip
;
;        symbol equates
;
sys=$9e             ;basic token for 'sys'
basic=$ff47         ;enter basic
readst=$ffb7        ;read io status
setlfs=$ffba        ;set logical file
setnam=$ffbd        ;set file name information
load=$ffd5          ;load file
hello=$0400         ;hello world location
 skip
;
;        zero page
;
         *=1
ebank    *=*+1
         *=$40
ldfadr   *=*+1      ;load file defn addr
 skip
```

Sample Part 2 – The program loader (LOADER.SOR) continued

```
;
;        basic header (10 sys2061)
;
         *=$0801
bas0     byt <bas1,>bas1  ;next line address
         byt 10,0         ;line number
         byt sys,'2061'   ;basic code
         byt 0            ;end of line marker
bas1     byt 0,0          ;end of program
 skip
;
;        main line (location 2061)
;
main01   lda #0      ;bank
         sta ebank   ;exec bank
         jsr load01  ;load program files
         bne main02  ;load failed
 skip
         jsr hello   ;invoke 'hello world'
 skip
main02   jmp basic   ;return to basic
 skip
;
;        load program files
;
load01   lda #<ldftab;load file table addr (low)
         sta ldfadr  ;load file defn addr (low)
         lda #>ldftab;load file table addr (high)
         sta ldfadr+1;load file defn addr (high)
 skip
load02   lda #1      ;logical file
         ldx #8      ;device
         tay         ;secondary address
         jsr setlfs  ;set logical file ($ffba)
         ldy #0
         lda (ldfadr),y ;file defn length
         beq load04     ;no more load files
 skip
;                    ;parm idx = defn length - 2
         sec
         sbc #2
         pha         ;save parm idx on stack
;                    ;fnlen = parm idx - 1
         sbc #1      ;file name length in (a)
;                    ;set filename address
         ldx ldfadr  ;load file defn addr (low)
         ldy ldfadr+1;load file defn addr (high)
         inx         ;filename addr (low)
         bne load03
 skip
         iny         ;filename addr (high)
```

Sample Part 2 – The program loader (LOADER.SOR) continued

```
load03   jsr setnam  ;set file name information ($ffbd)
         pla         ;restore parm idx from stack
         tay
;                    ;get the start address for load
         lda (ldfadr),y ;<start
         pha
         iny
         lda (ldfadr),y ;>start
         tax         ;<start
         pla
         tay         ;>start
         lda #0      ;load flag
         jsr load    ;load the file
         jsr readst  ;read io status
         and #$bf    ;ignore eof status
         bne load04  ;load failed, terminate
;                    ;advance to next load file
         clc
         ldy #0
         lda (ldfadr),y ;file defn length
 skip
         adc ldfadr  ;load file defn addr (low)
         sta ldfadr  ;load file defn addr (low)
         bcc load02
 skip
         inc ldfadr+1;load file defn addr (high)
         bcs load02
 skip
load04   rts         ;return
 skip 3
;
;        load file table
;
;                     ;hello world
ldftab   byt z0fln    ;filename length
         byt 'helloworld.prg' ;filename
         byt <hello,>hello ;start address
z0fln=*-ldftab
 skip
ldftaben byt 0        ;end of table marker
 skip
         end
```

SAMPLE LISTINGS

HELLO WORLD (HELLOWORLD.LST)

```
PAGE  0001              HELLOWORLD.SOR       DATE  09/08/23

LINE  LOC   CODE        SOURCE

0001  0000              ;
0002  0000              ;        HELLO WORLD SUB-ROUTINE
0003  0000              ;
0004  0000              PRINT=$FFD2
0005  0000                       *=$0400
0006  0400  A2 00                LDX #0
0007  0402  BD 0E 04    LOOP     LDA HELLO,X
0008  0405  F0 06                BEQ EXIT
0010  0407  20 D2 FF             JSR PRINT
0011  040A  E8                   INX
0012  040B  D0 F5                BNE LOOP
0014  040D  60          EXIT     RTS
0016  040E  21 21 21 20 HELLO    BYT '!!! HELLO WORLD !!!'
0017  0421  00                   BYT 0
0018  0422                       END


ERRORS = 0000

SYMBOL TABLE

SYMBOL   VALUE

 EXIT     040D    HELLO    040E    LOOP     0402    PRINT    FFD2

END OF ASSEMBLY
```

LOADER (LOADER.LST)

```
PAGE  0001              LOADER.SOR           DATE  09/08/23

LINE  LOC   CODE        SOURCE

0001  0000              ;******************************************;
0002  0000              ;                                          ;
0003  0000              ;        PROGRAM LOADER                    ;
0004  0000              ;                                          ;
0005  0000              ;******************************************;
0007  0000              ;
0008  0000              ;        SYMBOL EQUATES
0009  0000              ;
0010  0000              SYS=$9E              ;BASIC TOKEN FOR 'SYS'
0011  0000              BASIC=$FF47          ;ENTER BASIC
0012  0000              READST=$FFB7         ;READ IO STATUS
0013  0000              SETLFS=$FFBA         ;SET LOGICAL FILE
0014  0000              SETNAM=$FFBD         ;SET FILE NAME INFORMATION
0015  0000              LOAD=$FFD5           ;LOAD FILE
0016  0000              HELLO=$0400          ;HELLO WORLD LOCATION
0018  0000              ;
0019  0000              ;        ZERO PAGE
0020  0000              ;
0021  0000                       *=1
0022  0001              EBANK    *=*+1
0023  0002                       *=$40
0024  0040              LDFADR   *=*+1       ;LOAD FILE DEFN ADDR
0026  0041              ;
0027  0041              ;        BASIC HEADER (10 SYS2061)
0028  0041              ;
0029  0041                       *=$0801
0030  0801  0B 08       BAS0     BYT <BAS1,>BAS1  ;NEXT LINE ADDRESS
0031  0803  0A 00                BYT 10,0         ;LINE NUMBER
0032  0805  9E 32 30 36          BYT SYS,'2061'   ;BASIC CODE
0033  080A  00                   BYT 0            ;END OF LINE MARKER
0034  080B  00 00       BAS1     BYT 0,0          ;END OF PROGRAM
0036  080D              ;
0037  080D              ;        MAIN LINE (LOCATION 2061)
0038  080D              ;
0039  080D  A9 00       MAIN01   LDA #0      ;BANK
0040  080F  85 01                STA EBANK   ;EXEC BANK
0041  0811  20 1C 08             JSR LOAD01  ;LOAD PROGRAM FILES
0042  0814  D0 03                BNE MAIN02  ;LOAD FAILED
0044  0816  20 00 04             JSR HELLO   ;INVOKE 'HELLO WORLD'
0046  0819  4C 47 FF    MAIN02   JMP BASIC   ;RETURN TO BASIC
0048  081C              ;
0049  081C              ;        LOAD PROGRAM FILES
0050  081C              ;
0051  081C  A9 6A       LOAD01   LDA #<LDFTAB;LOAD FILE TABLE ADDR (LOW)
0052  081E  85 40                STA LDFADR  ;LOAD FILE DEFN ADDR (LOW)
0053  0820  A9 08                LDA #>LDFTAB;LOAD FILE TABLE ADDR (HIGH)
0054  0822  85 41                STA LDFADR+1;LOAD FILE DEFN ADDR (HIGH)
0056  0824  A9 01       LOAD02   LDA #1      ;LOGICAL FILE
0057  0826  A2 08                LDX #8      ;DEVICE
0058  0828  A8                   TAY         ;SECONDARY ADDRESS
0059  0829  20 BA FF             JSR SETLFS  ;SET LOGICAL FILE ($FFBA)
0060  082C  A0 00                LDY #0
```

LOADER (LOADER.LST) continued

```
PAGE  0002               LOADER.SOR           DATE  09/08/23

LINE  LOC   CODE         SOURCE

0061  082E  B1 40                 LDA (LDFADR),Y ;FILE DEFN LENGTH
0062  0830  F0 37                 BEQ LOAD04     ;NO MORE LOAD FILES
0064  0832               ;                  ;PARM IDX = DEFN LENGTH - 2
0065  0832  38                    SEC
0066  0833  E9 02                 SBC #2
0067  0835  48                    PHA         ;SAVE PARM IDX ON STACK
0068  0836               ;                  ;FNLEN = PARM IDX - 1
0069  0836  E9 01                 SBC #1      ;FILE NAME LENGTH IN (A)
0070  0838               ;                  ;SET FILENAME ADDRESS
0071  0838  A6 40                 LDX LDFADR  ;LOAD FILE DEFN ADDR (LOW)
0072  083A  A4 41                 LDY LDFADR+1;LOAD FILE DEFN ADDR (HIGH)
0073  083C  E8                    INX         ;FILENAME ADDR (LOW)
0074  083D  D0 01                 BNE LOAD03
0076  083F  C8                    INY         ;FILENAME ADDR (HIGH)
0077  0840  20 BD FF     LOAD03   JSR SETNAM  ;SET FILE NAME INFORMATION ($FFBD)
0078  0843  68                    PLA         ;RESTORE PARM IDX FROM STACK
0079  0844  A8                    TAY
0080  0845               ;                  ;GET THE START ADDRESS FOR LOAD
0081  0845  B1 40                 LDA (LDFADR),Y ;<START
0082  0847  48                    PHA
0083  0848  C8                    INY
0084  0849  B1 40                 LDA (LDFADR),Y ;>START
0085  084B  AA                    TAX         ;<START
0086  084C  68                    PLA
0087  084D  A8                    TAY         ;>START
0088  084E  A9 00                 LDA #0      ;LOAD FLAG
0089  0850  20 D5 FF              JSR LOAD    ;LOAD THE FILE
0090  0853  20 B7 FF              JSR READST  ;READ IO STATUS
0091  0856  29 BF                 AND #$BF    ;IGNORE EOF STATUS
0092  0858  D0 0F                 BNE LOAD04  ;LOAD FAILED, TERMINATE
0093  085A               ;                  ;ADVANCE TO NEXT LOAD FILE
0094  085A  18                    CLC
0095  085B  A0 00                 LDY #0
0096  085D  B1 40                 LDA (LDFADR),Y ;FILE DEFN LENGTH
0098  085F  65 40                 ADC LDFADR  ;LOAD FILE DEFN ADDR (LOW)
0099  0861  85 40                 STA LDFADR  ;LOAD FILE DEFN ADDR (LOW)
0100  0863  90 BF                 BCC LOAD02
0102  0865  E6 41                 INC LDFADR+1;LOAD FILE DEFN ADDR (HIGH)
0103  0867  B0 BB                 BCS LOAD02
0105  0869  60           LOAD04   RTS         ;RETURN
0107  086A               ;
0108  086A               ;        LOAD FILE TABLE
0109  086A               ;
0110  086A               ;                   ;HELLO WORLD
0111  086A  11           LDFTAB   BYT Z0FLN    ;FILENAME LENGTH
0112  086B  48 45 4C 4C           BYT 'HELLOWORLD.PRG' ;FILENAME
0113  0879  00 04                 BYT <HELLO,>HELLO ;START ADDRESS
0114  087B               Z0FLN=*-LDFTAB
0116  087B  00           LDFTABEN BYT 0        ;END OF TABLE MARKER
0118  087C                        END
```

LOADER (LOADER.LST) continued

```
PAGE  0003              LOADER.SOR           DATE  09/08/23

LINE  LOC   CODE        SOURCE

ERRORS = 0000

SYMBOL TABLE

SYMBOL   VALUE

 BAS0     0801    BAS1     080B    BASIC    FF47    EBANK    0001
 HELLO    0400    LDFADR   0040    LDFTAB   086A    LDFTABEN 087B
 LOAD     FFD5    LOAD01   081C    LOAD02   0824    LOAD03   0840
 LOAD04   0869    MAIN01   080D    MAIN02   0819    READST   FFB7
 SETLFS   FFBA    SETNAM   FFBD    SYS      009E    Z0FLN    0011

END OF ASSEMBLY
```

APPENDIX C

WILD CARD FEATURE

ASMEDIT has the capability to display a list of source files, selected from the current directory, from which a program or program segment may be selected and loaded into the Edit buffer. This is achieved using the 'wild card' character '?'.

The number and positions of the '?'s, interspersed within the code name and segment input areas, specifies the criteria to dictate whether the file is to be displayed in the resulting list.

Here is an example to illustrate this capability and how it can be used.

Suppose a number of code files exist as follows:-

```
'1 APPLE.SOR'
'2 APPLE.SOR'
'1 PINEAPPLE.SOR'
'2 PINEAPPLE.SOR'
'1 APPLE PIE.SOR'
```

Let us now suppose that we wish to display, in our subsequent list, all files ENDING in the letters APPLE then segment and program name should be specified as follows:-

```
SEGMENT      ?
CODE NAME    ?APPLE
```

The resulting display would list:

```
'1 APPLE.SOR'
'2 APPLE.SOR'
'1 PINEAPPLE.SOR'
'2 PINEAPPLE.SOR'
```

Note that '1 APPLE PIE.SOR' is not selected because although the file name contains the letters APPLE the file name does not actually end in APPLE.

APPENDIX D

EDITOR CONDITION CODES

Code	Explanation	Action
1	The space allocated to the Edit Buffer has been exceeded.	Reset with ESC Key. Deleted last line or lines processed. To continue save code so far and clear the buffer.
	If this occurs with the Verify command then the space allocated to the Symbol buffer has been exceeded.	Reset with ESC Key. Reduce the number of symbols and/or symbol name lengths.
2	The character string searched for does not exist in the Edit buffer.	Reset with ESC Key.
3	The last parameter entered has illegal content.	Reset with ESC Key. Check the permissible parameter contents under the appropriate command description.
4	This isn't an error situation. The last command invoked has ended successfully.	Reset with ESC Key.

APPENDIX D continued

MENU ERROR MESSAGES

Error Message	Action
MEMORY EXCEEDED	If occurs during editing then attempt to re-enter the Editor and delete text as necessary. If re-entry is not allowed there is no alternative but to re-load from disk or clear the buffer through MENU 1. If occurs during Assembly then the area allocated to symbols has been exceeded in which case look to reduce the number of symbols or the symbol name lengths.
INVALID INPUT	One or more Input Areas in the menu have incorrect values. The inputs in error are highlighted with an adjacent '?' in reverse field. Correct and press RETURN again.
O.K.	NOT an error. Successful end.
LOAD ERROR	Try again. This message is often supplemented with an explanatory DOS message - i.e. (FILE NOT FOUND).
INVALID FILE	You are attempting to load a file which is not ASMEDIT source code or is corrupt.
SAVE ERROR	Try again. This message is often supplemented with an explanatory DOS message - i.e. (FILE EXISTS).
OPEN FILE ERROR	The editor cannot open a source code file. A supplementary DOS error message should be provided giving reason for failure.
SOURCE FILE ERROR	The assembler cannot open a source code file during assembly. A supplementary DOS message should be provided giving reason for failure.
CODE FILE ERROR	The assembler cannot open a file to receive the code generated during assembly. A supplementary DOS error message should be provided giving reason for failure.
LIST FILE ERROR	The assembler cannot open the disk output file or printer during assembly. For a disk file a supplementary DOS error message should be provided giving reason for failure. If attempting to print the listing then no printer has been detected.
DOS ERROR	A DOS command has failed. A supplementary DOS error message should be provided giving reason for failure.

APPENDIX E

EDITOR COMMANDS QUICK REFERENCE

Commands in Scan/Edit (Mode S)

Commands are entered starting at the command prompt '==>' at the top of the display. The First character entered is always the command. Some commands are terminated with [ESC].

Find String	[F] { *string* { [RETURN] } } [ESC]
Alter String	[A] *old-str* [RETURN] *new-str* [RETURN] ([A] [RETURN]) / ([F] { [RETURN] }) [ESC]
Delete Lines	[D] *start* [RETURN] *end* [RETURN]
Copy Lines	[C] *start* [RETURN] *end* [RETURN] { *insert-after* [RETURN] } [ESC]
Move Lines	[M] *start* [RETURN] *end* [RETURN] *insert-after* [RETURN]
Transcribe Lines	[T] *start* [RETURN] *end* [RETURN] { *insert-after* [RETURN] } [ESC]
Copy Column	[SHIFT]+[C] *top-left* [RETURN] *bottom-right* [RETURN] { *to* [RETURN] } [ESC]
Move Column	[SHIFT]+[M] *top-left* [RETURN] *bottom-right* [RETURN] *to* [RETURN]
Transcribe Column	[SHIFT]+[T] *top-left* [RETURN] *bottom-right* [RETURN] { *to* [RETURN] } [ESC]
Copy Section	[ALT]+[C] *section* [RETURN] { { [PGUP] or [PGDN] } [RETURN] } [ESC]
Move Section	[ALT]+[M] *section* [RETURN] { [PGUP] or [PGDN] } [RETURN]
Transcribe Section	[ALT]+[T] *section* [RETURN] { { [PGUP] or [PGDN] } [RETURN] } [ESC]
Insert Section	[I] *insert-after* [RETURN]
Delete Section	[ALT]+[D] *section* [RETURN]
Erase Section	[SHIFT]+[E] *section* [RETURN]
Goto Section	[G] *section* [RETURN]
Undo Section	[U] [RETURN]
Previous Section	{ [PGUP] }
Next Section	{ [PGDN] }
Key Mode	[F3] *input* [ESC]
Exit Scan Edit	[X]
Save and Exit Scan Edit	[SHIFT]+[X]
Swap Buffers	[S]
Quick Reference	[F1]

Commands in Input Mode (Overwrite - 'O' or insert 'I')

Insert/overwrite	[INST]	
Insert new line	[^]	(Repeats)
Delete character (BWD)	[Backspace]	(Repeats)
Delete character (FWD)	[DEL]	(Repeats)
Set/Unset Tab	[CTRL]+[TAB]	
Display option	[F12]	
'S' Mode	[ESC]	
Next tab position	[TAB]	(Repeats)
Previous tab position	[SHIFT]+[TAB]	(Repeats)