intel®

AEDIT-86 TEXT EDITOR USER'S GUIDE

Order Number: 121756-003

AEDIT-86 TEXT EDITOR USER'S GUIDE

Order Number: 121756-003

BITBUS	i_m	iSBC	Plug-A-Bubble
COMMputer	iMMX	iSBX	PROMPT
CREDIT	Insite	iSDM	Promware
Data Pipeline	$int_e l$	iSXM	QueX
Genius	int_elBOS	KEPROM	QUEST
i	Intelevision	Library Manager	Ripplemode
$\hat{i}$	int_eligent Identifier	MCS	RMX/80
I^2ICE	int_eligent Programming	Megachassis	RUPI
ICE	Intellec	MICROMAINFRAME	Seamless
iCS	Intellink	MULTIBUS	SOLO
iDBP	iOSP	MULTICHANNEL	SYSTEM 2000
iDIS	iPDS	MULTIMODULE	UPI
iLBX	iRMX		

A1344 / 385 / 6K / DD / KH
SOFTWARE

REV.	REVISION HISTORY	DATE	APPD.
-001	Original issue.	1/83	
-002	Includes additional macro files for non-Intel terminals.	5/83	
-003	Corrects bugs; adds PARAGRAPH, WINDOW, and KILL_WND commands; adds CALC and local and global variables; adds features to SET command; adds invocation controls; enlarges OTHER buffer.	12/84	M.K.S.

PREFACE

This manual provides instructions for using the AEDIT-86 interactive text editor. AEDIT-86 can be run within several environments including Series III, Series IV, and iRMX operating system. Basic knowledge of your system (ISIS and RUN, iNDX or iRMX), is assumed. Specific information on running AEDIT-86 on different systems is given in the appendixes.

This manual is for both new users and those who are already familiar with microcomputers and text editors.

The manual is organized in two parts as follows:

Part I provides the basic information a user needs to use the AEDIT-86 text editor. It includes a tutorial, edit command descriptions, and a description of the AEDIT-86 invocation.

- Chapter 1, "Introduction and Tutorial," defines the hardware and software necessary to run AEDIT-86, describes invoking and leaving the editor, and provides a brief tutorial session.
- Chapter 2, "The Editor Basics," describes concepts essential to using AEDIT-86 and provides an in-depth description of the AEDIT-86 display.
- Chapter 3, "Editing Commands," describes editing commands. Commands are ordered in the chapter by subjects and without forward references (as much as possible). AEDIT-86 commands not mentioned in this chapter are described in Part II.
- Chapter 4, "AEDIT-86 Invocation," describes the invocation command syntax and the invocation controls.

Part II lists more advanced uses of AEDIT-86 and describes the AEDIT-86 string, numeric, local, and global variables, the macro-related commands—Macro and Execute, and the CALC command.

- Chapter 5, "Macro Commands," describes commands that allow you to create and use macros.
- Chapter 6, "AEDIT-86 Variables," describes the local and global AEDIT-86 variables you can access.
- Chapter 7, "CALC Command," describes the CALC command and illustrates it with examples. Arithmetic and logical operations are supported.
- Chapter 8, "Advanced AEDIT-86 Usage," demonstrates the open-ended approach of AEDIT-86. Macros that enable advanced functions like word processing are described in this chapter. The macro file USEFUL.MAC, that is supplied with AEDIT-86, is listed and discussed.
- Chapter 9, "Configuration Commands," describes and lists the configuration commands required to create an AEDIT-86 configuration file.
- Appendix A, "AEDIT-86 Command Summary," lists AEDIT-86 commands and command formats, and provides a brief explanation of each command.
- Appendix B, "AEDIT-86 Error Messages," lists AEDIT-86 invocation, editing, CALC, and macro error messages, their probable causes and how they affect the system.
- Appendix C, "Using AEDIT-86 on the Series III," describes the system-dependent usage of AEDIT-86 on a Series III.

- Appendix D, "Using AEDIT-86 on the Series IV," describes the system-dependent usage of AEDIT-86 on a Series IV.
- Appendix E, "Using AEDIT-86 on iRMX Based System," describes the system-dependent usage of AEDIT-86 on iRMX based systems.
- Appendix F, "Configuring AEDIT-86 for Other Terminals," describes how to configure AEDIT-86 for several terminals and provides tested configurations for specific terminals.
- Appendix G, "ASCII Codes," lists ASCII codes with their hexadecimal values.

Related Publications

References to specific operating systems are given in the appendixes.

Notational Conventions

`UPPERCASE`	Characters shown in uppercase must be entered in the order shown. You may enter the characters in uppercase or lowercase.
italics	Italics indicate variable information, such as filename.
[ ]	Brackets indicate optional arguments or parameters.
|	The vertical bar separates options within brackets.
...	Ellipses indicate that the preceding argument or parameter may be repeated.
punctuation	Punctuation other than ellipses, braces, and brackets must be entered as shown. For example, the punctuation shown in the following command must be entered: `RUN AEDIT.86 :F1:EXAMPL.SRC MACRO(WORDP.MAC)`
`input lines`	In interactive examples, input lines and user responses are printed in white on black to differentiate them from system output.
‹ ›	Angle brackets indicate a key configured for a function, e.g., ‹HEX› or ‹ESC›.

CONTENTS

PART I

FIGURES

TABLES

PART I

Part I includes Chapters 1 through 4. It gives a general overview of AEDIT-86 and describes the basics required to use AEDIT-86, including a tutorial. All AEDIT-86 commands are described in Part I, most of them in depth. However, some are mentioned briefly here, then fully described in Part II. Read Part I in the order it is presented before going on to Part II.

Chapter 1, "Introduction and Tutorial," introduces AEDIT-86 and gives a short tutorial. It describes activating the editor; entering, changing, deleting, and copying text; some function keys; the main and the OTHER files; and exiting the editor.

Chapter 2, "The Editor Basics," describes several basic AEDIT-86 features, including the keyboard, the cursor, the end-of-file marker, screen-mode editing, display and menu format, beep warning, lines and line terminators, printing and nonprinting characters, tags, repeat function, and AEDIT-86's three buffers.

Chapter 3, "Editing Commands," describes AEDIT-86's editing modes, commands, and function keys.

Chapter 4, "AEDIT Invocation," describes AEDIT-86 invocation and the controls FORWARDONLY, VIEWONLY, RECOVER, MACRO, MACROSIZE, and BATCH.

CHAPTER 1
INTRODUCTION AND TUTORIAL

1.1 Introduction

AEDIT-86 is an interactive, screen-oriented text editor with menu style command prompts. AEDIT-86 can be run within several environments including Series III, Series IV and iRMX Operating System. Specific information on running AEDIT-86 on the different systems is given in the appendixes. AEDIT-86 requires a minimum of 96 Kbytes of free RAM. Additional free RAM improves AEDIT-86's performance significantly.

AEDIT-86 takes advantage of CRT capabilities to allow you to:

- Display and scroll text on the screen
- Move to any position in the text file or to any point on the screen
- Rewrite text by typing new characters over old ones
- Make insertions and deletions easily
- Use windowing to view two files or two portions of the same file simultaneously

To simplify text editing, AEDIT-86 also provides features that allow you to:

- Find any string of characters
- Substitute one string of characters for another string
- Move or copy sections of text in a file or to another file
- Create macros to execute several commands at once, thereby simplifying repetitive editing tasks
- Use arithmetic functions
- Edit two files simultaneously
- Format text automatically
- Justify paragraphs
- View lines over 80 characters long
- Use basic word processing operations

1.2 AEDIT-86 Tutorial

This session is a short tutorial that illustrates the most basic AEDIT 86 commands. The following functions are covered:

- Activating the editor
- Entering text
- Changing text
- Deleting text
- Copying text
- Using the OTHER Command
- Exiting the editor

The purpose of this tutorial is to get you started, not to fully document AEDIT-86 commands. Only a few of the most basic AEDIT-86 commands are presented in this tutorial. For more complete information on AEDIT-86 features, concepts and commands, read Chapters 1 through 4.

This manual makes no references to any particular operating system; the appendixes give operating-system dependent information.

1.2.1 Activating the Editor

Specific invocation commands are given in the appendixes.

In general, AEDIT-86 is activated by typing the following:

```
-EDIT <RETURN>
```

The editor displays the following prompt at the bottom of the screen:

```
-??- system-id AEDIT Vx.y Copyright yyyy Intel Corp.
Again   Block   Calc   Delete   Execute   Find   -find   --more--
```

The question marks (-??-) in front of the *system-id* indicate that AEDIT-86 is "waiting" for your input. When AEDIT-86 is "busy," the question marks are replaced by two exclamation points (-!!-). *system-id* is a string identifying the operating system, *x.y* is the AEDIT-86 version number, and *yyyy* is the copyright year(s). The vertical bar (|) (initially in the upper left corner of the screen) marks the end of the file (EOF). Because the new file you just created has no text, the vertical bar appears at the top left corner of the display area. As you type text into the file, the vertical bar moves and continues to mark the end of the file. The cursor initially covers the EOF marker. (The term cursor refers to the position indicator. The way in which the cursor is displayed—solid, non-blinking block, underline, etc.—is terminal dependent.)

When first invoked, AEDIT-86 is at main command level waiting for your input. The menu prompt line offers a selection of main commands or modes (XCHANGE and INSERT are considered modes). When invoked, several main commands offer subcommands. You must be at main command level to execute commands, except when using cursor movement commands and the delete keys. AEDIT-86 does not return automatically to main command level after executing some commands (e.g., FIND). To return to the main command level or to exit INSERT or XCHANGE modes, press <ESC>.

Throughout the manual, main commands and modes are written in uppercase to distinguish them from subcommands, which are written in upper/lowercase; e.g., QUIT Exit. Function keys are enclosed in angle brackets; e.g., <ESC>, <HEX>.

To specify a menu selection (i.e., command or mode), press the initial letter of the selection (e.g., B for BLOCK).

The word --more-- on the prompt line indicates that there are more commands or modes. Press <TAB> to display the next line of prompts. Pressing <TAB> to display additional prompts is a cyclic operation (i.e., after the last prompt line is displayed, the first is redisplayed).

1.2.2 Entering, Changing, and Deleting Text

Before typing text into the file, you must press I to enter INSERT mode. The word [insert] is displayed at the bottom of the screen, indicating that you are in INSERT mode. Type a word but misspell it. To correct the error, press <RUBOUT>. Each time you press <RUBOUT>, the cursor backs up one column and erases the character. When the erroneous character(s) is erased, type the correct character(s).

The line you just typed may be deleted character-by-character with the ‹RUBOUT› key or in its entirety with the key configured to ‹DELLI›, delete line (usually configured to ‹CTRL-Z›). Delete the line. The file is now empty, and the EOF marker is back in the upper left-hand corner of the screen. The cursor, however, remains in the same position on the screen until the next command is given.

Now type the following sentence. Enter the text exactly as shown—several words are deliberately misspelled.

```
High-levell languages (Pacal in particular) more <RETURN>
closely modal the human thought process than low-level<RETURN>
languages such as assembly language.<RETURN>
```

The first word in the sentence, *levell*, is misspelled. To correct this error, use the cursor control keys to position the cursor "over" the erroneous *l*. The cursor is moved by the cursor control keys (arrows), in the direction indicated by the arrow. (Therefore, press the up arrow (‹UP›) twice to move the cursor to the first line. Then press the left arrow ‹LEFT› followed by ‹HOME›—the ‹HOME› key is used in conjunction with the cursor control keys for fast cursor movement—to move the cursor to the first position in the line. Now press the right arrow ‹RIGHT› nine times to position the cursor "on" the first *l*.) Then press the key configured to ‹DELCH›, delete character (usually configured to ‹CTRL-F›) to delete the extra *l*.

The *s* in *Pascal* has been omitted. To correct this error, position the cursor on the *c* in *Pascal* and type *s*. Text automatically moves to the right as the *s* is inserted.

Press ‹ESC› to leave INSERT mode and return to main command level.

The word *model* is misspelled *modal*. To correct the error, type X to enter XCHANGE mode.

The word [exchange] is displayed at the bottom of the screen, indicating that you are in XCHANGE mode. Position the cursor "over" the *a* and type *e*.

Press ‹ESC› to leave XCHANGE mode and return to main command level.

You have learned how to insert text, exchange text, and delete individual characters. Now type the following sentence exactly as shown. First, type I. Then, position the cursor below the lines you just typed. Move the cursor to the end of the line using the right arrow key (‹RIGHT›) followed by ‹HOME›, then press ‹DOWN› twice and ‹RETURN› once. The cursor is now positioned at the end of the file and at the beginning of an empty line. Type the following lines exactly as shown:

```
Thus, high-level languages are easier and faster to <RETURN>
write than low-level languages, since one less less <RETURN>
translation step is required from concept to code.<RETURN>
```

Press ‹ESC› to leave INSERT mode and return to main command level.

Note that the word *less* is typed twice. To correct this error position the cursor on the *l* of the second *less*. Because it appears at the end of the line, it may be deleted with ‹DELR›. Press the key configured to ‹DELR›, delete right (usually configured to ‹CTRL-A›). The ‹DELR› command deletes all text to the right of the cursor.

Suppose you wish to delete the phrase *from concept to code* from the text, leaving the period at the end of the sentence. To do this, you must "block" (i.e., delimit) this section from the rest of the text using the BLOCK command followed by the Delete subcommand.

First, position the cursor over the first character of the section. In this case you want the period to close the sentence, so position the cursor on the space before the *f* in *from.* Then press B for BLOCK. The @ sign covers the space. Then position the cursor one character past the end of the section you wish to block; i.e., place the cursor on the period immediately after the *e* in *code*. When you pressed B for BLOCK, the menu displayed several alternative subcommands: Buffer, Delete, Find, -find, Jump, and Put.

To delete the phrase, press D for Delete.

The phrase is deleted from the text and the space is closed automatically. The result is as follows:

```
High-level languages (Pascal in particular) more
closely model the human thought process than low-level
languages such as assembly language.
Thus, high-level languages are easier and faster to
write than low-level languages, since one less
translation step is required.
```

For faster cursor movement, use the (-)FIND or JUMP command. In the example above, to move the cursor to the word *human* (assuming that the current cursor position is at the end of the file), press the hyphen (-). AEDIT-86 prompts for the name of the target string. Type *human* and press < ESC >. The cursor moves to the *h* in *human*. To move forward in the file with the FIND command, press F, then type the "target string" followed by < ESC >. Note that < ESC > must be pressed to terminate the -FIND command.

To jump to the beginning or the end of the file, simply press J for JUMP followed by S for Start or E for End.

1.2.3 Copying Text

Use the BLOCK command to copy existing text. If you want to copy a section of text to another part of your file, delimit the text using the BLOCK command and press B to specify the Buffer subcommand. The text is held temporarily in a buffer. Then, position the cursor where you want the text to appear and press G, the GET command. This command prompts for an input file. Pressing < RETURN > "gets" the contents of the buffer (where the text had been held temporarily) and places it at the current cursor position.

To move a section of text to another part of your file and delete it from its present position, delimit the text using the BLOCK command and press D to specify the Delete subcommand. The text is held temporarily in a buffer. Then, position the cursor where you want the text to appear and press G, the GET command. The command prompts for an input file. Press < RETURN > to place the buffer contents at the current cursor position. Note that copying or deleting a section of text is controlled by either the Buffer or Delete subcommand under the BLOCK command.

To copy a section of text to another file, delimit the text using the BLOCK command and press P to specify the Put subcommand. The menu prompts for an output file. Type in the filename and press < ESC > or < RETURN >. If the specified file already exists, the message "overwrite existing file? (y or [n])" is displayed. The file is copied only if you respond with *y*. If the specified file does not exist, it is created, and the text is copied to the specified file.

1.2.4 Using the OTHER Command

AEDIT-86 has two distinct and equivalent files: the main file and the OTHER file. The OTHER file increases the power of AEDIT-86, allowing you to switch between two editing files. To enter the OTHER file, use the OTHER command. Press O to enter the OTHER file. Press O a second time to return to the main file. The OTHER command is fully described in Chapter 3.

1.2.5 Exiting the Editor

To exit from the editor, press Q for QUIT.

The following prompt appears:

```
-??- no input file
Abort        Init        Write
```

Three alternative subcommands are available. A aborts the session; W saves the file (Init is described in Chapter 3).

If you do not want to save the contents of this practice session, press A to abort the session.

The following prompt is displayed:

```
all changes lost?  (y or [n])
```

Press *y*. The editor exits, and the file is not saved. Control returns to the operating system.

If you want to save this file, press W. The following prompt is displayed:

```
Output file:
```

Type in a filename (e.g., MYFILE) and press < RETURN >. The editor writes the specified file to the main storage device.

CHAPTER 2
THE EDITOR BASICS

The following editor basics are described in this chapter:

- Keyboard
- Cursor
- End-of-file (EOF) marker
- Screen-mode editing
- AEDIT-86 display and menu format
- Beep warning
- Lines and line terminators
- Printing and nonprinting characters
- Tags
- Repeat function (count)
- Buffer

2.1 Keyboard

The keyboard is your interface with the editor. It is a typewriter style, electronic keyboard that supports the ASCII character set.

Every keyboard character can be considered a command because every key causes something to happen. Most keys are self-explanatory. Some, however, are configured to perform functions rather than enter characters. These keys are called function keys; their names are enclosed in angle brackets throughout this manual.

ARROWS	The four keys labeled with directional arrows are the cursor control keys ‹LEFT›, ‹RIGHT›, ‹UP›, and ‹DOWN›.
CAPS LOCK or TPWR	The CAPS LOCK or TPWR key provides uppercase or lowercase entry of alphabetic characters. This key functions with the alphabetic keys only.
CONTROL ‹CTRL›	The CONTROL (‹CTRL›) key changes the function of some keys on the keyboard. (For example, to change the function of X, hold down the ‹CTRL› key and press X).
‹ESC›	The ‹ESC› (escape) key exits modes, terminates commands, and returns the editor to main command level.
‹HOME›	The ‹HOME› key allows faster cursor movement. Press an arrow key followed by ‹HOME› to page backward or forward through a file or to move rapidly to the beginning or end of a line. ‹HOME› is also used to enter the reedit mode for line-edit prompts.
‹RETURN›	The ‹RETURN› key moves the cursor to the beginning of the next line in INSERT and XCHANGE modes and at main command level. It also terminates line-edit prompt except for the search commands (-)FIND and (?)REPLACE. The carriage return/line feed character is displayed in text as a blank.

‹ RUBOUT ›	The RUBOUT key deletes the character to the left of the cursor at main command level or INSERT mode. In XCHANGE mode, ‹ RUBOUT › replaces the new character to the left of the cursor with the original character.
‹ TAB ›	The TAB key rotates the menu prompt line to display the next line of commands. In INSERT or XCHANGE modes, ‹ TAB › inserts blanks to the next defined tab stop (the default is every fourth column).

2.2 Cursor

The cursor indicates the entry point for all information. The cursor may be displayed as a solid nonblinking block, as an underline, as a solid blinking block, etc., depending on your terminal.

The term cursor refers both to the cursor displayed on the screen (the physical cursor) and to the current file location (the character "under" the cursor, or logical cursor). The physical and logical cursor diverge only when the physical cursor leaves the text area for prompt lines (for example, to enter a search command) or when the ‹ UP › or ‹ DOWN › command moves the cursor to a screen location past the end of a line. When the cursor is located past the end of a line, all commands except the ‹ UP › and ‹ DOWN › commands (including ‹ TAB › or an illegal command) move the cursor back to the end of the line, before execution.

The actual cursor position is the space between the character on which it appears and the character immediately to the left. Thus, for some commands (e.g., INSERT, BLOCK, ‹ DELL ›, ‹ DELR ›), characters are inserted (or deleted) directly *before* the character under the physical cursor. Some commands (e.g., XCHANGE and ‹ DELCH ›) operate on the character immediately to the right of the actual cursor position; other commands (e.g., ‹ RUBOUT ›) operate on the character immediately to the left of the actual cursor position.

2.3 End-of-File Marker

The end-of-file (EOF) marker is a vertical bar (|) that indicates the end of a file. If the end of a file is on the screen, the EOF marker is displayed immediately after the last character of the file.

2.4 Screen-Mode Editing

AEDIT-86's greatest advantage is its ability to display and verify changes to the text as you make them. You can move through your file making changes, insertions, and deletions, verifying them as you go. You can examine a screenful of text, locate the text you want to change, change it, and review the next screenful of text. A portion of text is always displayed on the screen.

2.5 AEDIT-86 Display/Menu Format

AEDIT-86 requires a CRT terminal (or a CRT section) with at least a 5-line, 80-column display screen (columns are numbered from 0 to 79). The cursor is the reference point for all operations, e.g., INSERT, FIND, DELETE, REPLACE. The screen is divided into the following three sections (listed from bottom up):

- Prompt line
- Message line
- Text area

Figure 2-1 shows the screen after AEDIT-86 is called but before any text has been typed.

2.5.1 The Prompt Line

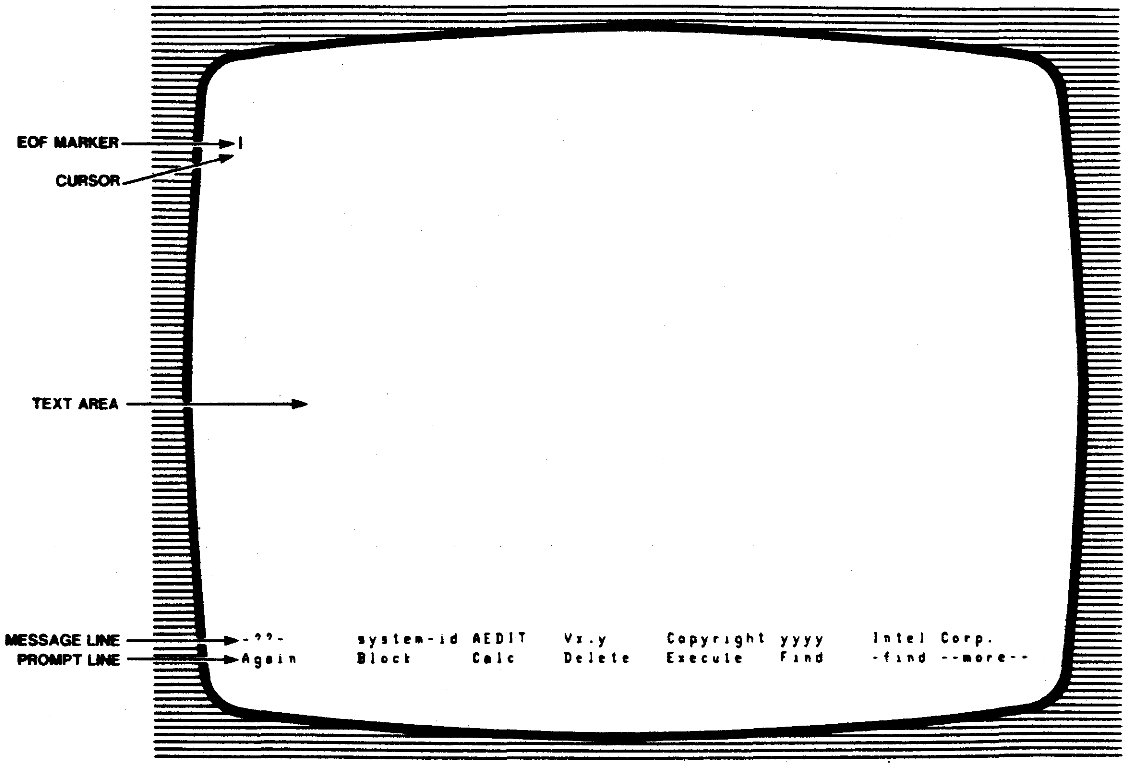
The prompt line is the bottom line of the display. (The *first position* of the prompt line is blank.) The prompt line contains information on the options of commands or subcommands that you may perform. The three types of prompts are

- Menu prompts
- Line-edited prompts
- Yes/no prompts

The Menu Prompt

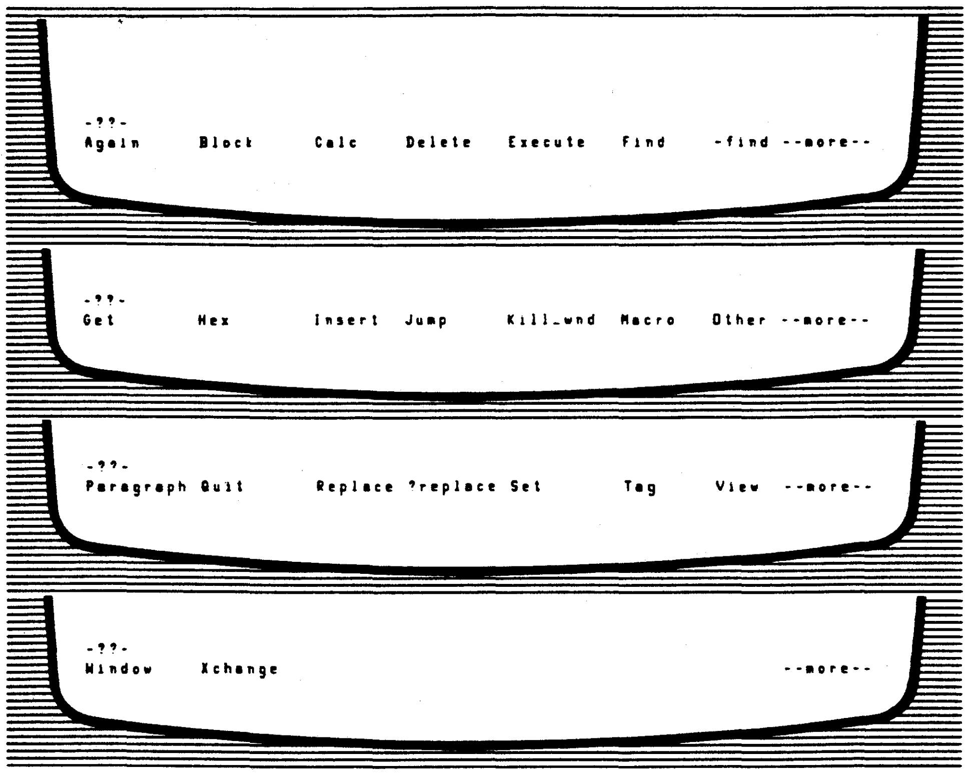
When AEDIT-86 is first invoked, the editor is at main command level and the menu prompt is displayed. Menu prompts are a partial list of up to eight words indicating available commands. The word --more-- indicates that pressing <TAB> displays the next line of prompts. The <TAB> command is cyclic (i.e., after the last prompt line is displayed the first is redisplayed). Figure 2-2 shows the four prompt lines available at main command level.

To select the desired command, type the first character of the prompt word (indicated as an uppercase letter). The prompt for a command does not have to be visible to invoke it—it may be on one of the prompt lines indicated by --more--. The prompt line goes blank at the beginning of the command operation and is restored at the end.



```
-??-        system-id AEDIT   Vx.y     Copyright yyyy    Intel Corp.
Again       Block     Calc    Delete   Execute   Find    -find --more--
```

Figure 2-1. AEDIT-86 Display 121756-1



```
-??-
Again    Block    Calc    Delete    Execute    Find    -find --more--
```

```
-??-
Get      Hex      Insert  Jump      Kill_wnd   Macro   Other --more--
```

```
-??-
Paragraph Quit    Replace ?replace Set        Tag     View  --more--
```

```
-??-
Window   Xchange                                            --more--
```

Figure 2-2. Menu Prompt Lines 121756-2

The Line-Edited Prompt

Line-edited prompts ask for information (such as a filename) that requires more than a single-character user response. The response can be up to 60 characters. It is terminated and the information sent by pressing < ESC >. In most cases < RETURN > may also be used to terminate a command. However, the search commands, (-)FIND and (?)REPLACE, must be terminated with the < ESC > key.

The first time a command is requested, the prompt line is empty and you simply enter the required information.

If the command (or a related command) has been requested previously, the prompt line contains the information entered at that time. This information may be reentered or edited, or new information may be entered.

If you want to use the previously entered information with no changes; e.g., to repeat a FIND, press < ESC > and this information will be reentered.

To edit the previously entered information, press < HOME > to enter reedit mode. The cursor may now be moved to the position you want to edit, e.g., to correct a typing error in a target-string. The delete key commands (< DELCH >, < DELLI >, < DELL >, < DELR >, < RUBOUT >), and cursor movement commands (< LEFT >, < RIGHT >, < HOME >) are legal line-edit commands. After entering the changes, press < ESC > to terminate line-edited input and send the entire string or press < RETURN > to terminate the string at the cursor. (The < ESC > key must be used for the search commands, (-)FIND and (?)REPLACE.) The characters to the left of the cursor are sent, those to the right of the cursor are lost.

If you want to enter new information, simply type it in. As soon as any key is pressed (except for <LEFT>, <RIGHT>, <HOME>, or <ESC>), the prompt line is blanked and the new characters are entered. Press <ESC> or <RETURN> to finish the command and send the information.

To enter a character using its ASCII value in the line-edit prompt, simply type <HEX> (usually configured as <CTRL-H>) followed by two hexadecimal digits. For example, <HEX>41 enters an A. This option allows you to enter control characters (e.g., <ESC>) into the text.

The Yes/No Prompt

The yes/no prompt form is "prompt ? (y or [n])", where square brackets surround the default, as in the following example:

```
all changes lost?  (y or [n])
```

In this case, n is the default; therefore, any response other than *y* (or *Y*) is considered a negative response. (In a yes/no prompt where *y* is the default, any response other than *n* (or *N*) is considered a positive response.)

2.5.2 Message Line

The message line is directly above the prompt line. It is used to display status messages or to indicate the command mode.

The busy/waiting indicator is displayed on the message line. For commands that take a relatively long time to execute, the busy/waiting indicator tells you if AEDIT-86 is still processing the command, or if it is ready to receive new input. This indicator is a configurable feature (described in Chapter 9). The busy/waiting indicator may be displayed as one of the following:

- -??- —indicates that the feature is on and AEDIT-86 expects input.
- -!!- —indicates that the feature is on and AEDIT-86 is executing a command.
- ---- —indicates that the feature has been turned off with an AEDIT-86 configuration command or that the feature is on, but the message line is for the nonactive window.

If the busy/waiting indicator feature is on, -??- is displayed whenever AEDIT-86 is waiting for input. The expected input may be a command, a subcommand, a yes/no answer, an operand, etc. Also, whenever AEDIT-86 is processing a command, -!!- is displayed. If the busy/waiting indicator feature is off, ---- is displayed at all times. Later on in this manual the busy/waiting indicator is displayed as four dashes.

If, for example, you type JS for JUMP Start, and the indicator is on, it is changed to -!!- as soon as AEDIT-86 accepts the command. The -!!- remains until AEDIT-86 completes the command execution. The indicator is then changed back to -??-.

Next, one of the following status words may be displayed:

- Macro — indicates that a macro is being defined.
- Other — indicates that the OTHER file is being edited.
- View — indicates that the VIEWONLY control is in effect.
- Forward — indicates that the FORWARDONLY control is in effect.

This part of the message line does not change unless the OTHER, MACRO, or editing mode of the file is changed. Other messages displayed on the message line are status messages, count (repeat function), and the line-edit prompt "<HOME> to re-edit".

AEDIT-86 does not write past the last column of the message line. If a message does not fit, ! is printed as the last character.

When the message line contains status information, usually an error message, the message line goes blank as soon as any key is pressed.

2.5.3 Text Area

The rest of the screen is the text area.

2.6 Beep Warning

The editor beeps when you try to do something illegal, for example:

- Attempting to execute an illegal command
- Typing an invalid character during INSERT or XCHANGE mode
- Typing more than 60 characters in a line-edited prompt
- Entering a repeat count greater than the maximum value

It also beeps when presenting some of the error messages.

2.7 Lines and Line Terminators

A line of text consists of a sequence of characters terminated by a carriage return/line feed. This pair of characters, called the line terminator, is entered in the file when you press < RETURN >. (RETURN is displayed on the screen as a blank.)

If a line is over 80 characters long, an exclamation point (!) is displayed in the last column on the screen. The portion of the line that does not fit on the screen is not displayed. To view the portion that is not displayed, use the SET Leftcol command (described in Chapter 3).

A line may contain any number of characters. AEDIT-86 breaks lines longer than 255 characters into 255-character segments. A plus sign (+) is displayed at the end of each segment.

2.8 Printing and Nonprinting Characters

In general, all characters except those with ASCII values under 20H and characters with hexadecimal values equal to or above 7FH are displayed on the screen. All characters that are not displayed on the screen print as a question mark (?). Carriage return and tab print as blanks. If the Highbit feature (described in Chapter 3) is set, characters with hexadecimal values over 7FH are displayed as is.

2.9 Tags

Tags identify locations in a file. You can specify four locations, A through D, with the TAG command and use them as destinations for the JUMP command. Tags are invisible and are not saved when you exit the file.

2.10 Repeat Function (Count)

Count is displayed on the message line and indicates the number of times to repeat a command. Some commands ignore count or, like delete character < DELCH >, limit count. Enter count before typing a command letter. It is then displayed at the left side of the message line. < RUBOUT > can be used to delete the value being entered for count. The cursor position after a command has been executed *count* times is its location when count is exhausted or no more occurrences are found. When the message line contains a count, the count is blanked when the next prompt is issued. The repeat count is an optional decimal repetition factor in the range 0 to 65535 (2**16-1). Any attempt to type a larger value for count causes AEDIT-86 to beep. A forward slash (/) is accepted as a count and means repeat forever. The default count is one.

2.11 Buffer

AEDIT-86 has three buffers: the main buffer, the OTHER buffer, and the Block buffer. All three buffers are allocated space in the user's free RAM.

The main buffer is the text area at startup. It always contains a portion of the main file.

The OTHER buffer is accessed with the OTHER command and always contains a portion of the OTHER file (if one exists).

The buffer that is accessed and active is referred to as the "current" buffer; the one that is not being edited, as the "secondary" buffer. For example, if you are editing a file in the OTHER buffer, it would be referred to as the current buffer, and the main buffer would be referred to as the secondary buffer.

If either the main or OTHER buffer is too small for the text file, AEDIT-86 extends the buffer with additional free RAM if it is available. When all free RAM is exhausted, AEDIT-86 writes to temporary files, usually on disk or diskette. AEDIT-86's performance improves with the amount of free RAM available.

The Block buffer is the storage area for text that you move, copy, or delete, using the BLOCK/DELETE commands. The Block buffer allows you to move text between the main and the OTHER file. The Block buffer has a fixed size of 2 Kbytes. If more than 2 Kbytes is required, AEDIT-86 uses a temporary file.

CHAPTER 3
EDITING COMMANDS

This chapter describes all AEDIT-86 commands. Most of the commands are described in depth. However, some commands are only mentioned here and fully described in Part II.

3.1 Cursor Movement Commands

The cursor movement commands control cursor movement in a file. <LEFT>, <RIGHT>, <UP>, and <DOWN> refer to keys labeled with directional arrows; <HOME> refers to the key labeled <HOME> (or configured for this function).

3.1.1 <LEFT>

The LEFT command moves the cursor one character to the left.

Special cases of <LEFT> are—

- If the cursor is on the first character of the file, the command is ignored.
- If the cursor is at the top of the screen and at the beginning of the line, the screen is rewritten/scrolled to display previous line(s) of text (the number of lines scrolled is terminal dependent).
- If the cursor is at the beginning of a line, the cursor moves to the last character of the previous line.
- If the cursor is moving to a tab, <LEFT> skips over the tab.

Count. This command accepts any count where count multiplies the distance moved. If count is / or greater than the number of characters that exist from the beginning of the file to the current cursor position, the cursor jumps to the beginning of the file. The result of a repeated move is not displayed until the move is complete.

Related Commands. <HOME>

3.1.2 <RIGHT>

The RIGHT command moves the cursor one character to the right.

Special cases of <RIGHT> are—

- If the cursor is on the EOF marker, the command is ignored.
- If the cursor is on the last character of the last text line on the screen, the screen scrolls up (usually one line).
- If the cursor is on the last character of a line, it moves to the beginning of the next line.
- If the cursor is moving to a tab, <RIGHT> skips over the tab.

Count. This command accepts any count where count multiplies the distance moved. If count is / or greater than the number of characters that exist in the file from the current cursor position to the end of the file, the cursor jumps to the end of the file. The result of a repeated move is not displayed until the move is complete.

Related Commands. <HOME>

3.1.3 ‹UP›

The UP command moves the cursor up one line in the same column.

Special cases of ‹UP› are—

- If the cursor is in the top line of the file, the command is ignored.
- If the cursor is in the top line of the screen, the screen is rewritten/scrolled to display previous line(s) of text. The cursor remains in the same column as before the command.

Count. This command accepts any count where count multiplies the distance moved. If count is / or greater than the number of lines that exist from the beginning of the file to the current cursor position, the cursor jumps to the first line of the file. The result of a repeated move is not displayed until the move is complete.

Related Commands. ‹HOME›

3.1.4 ‹DOWN›

The DOWN command moves the cursor down one line in the same column.

Special cases of ‹DOWN› are—

- If the cursor is on the bottom line of the file, the command is ignored.
- If the cursor is in the last line of text on the screen, the screen scrolls up. The cursor remains in the same column as before the command.

Count. This command accepts any count where count multiplies the distance moved. If count is / or greater than the number of lines that exist in the file from the current cursor position to the end of the file, the cursor jumps to the last line of the file. The result of a repeated move is not displayed until the move is complete.

Related Commands. ‹HOME›

3.1.5 ‹HOME›

The HOME command has two functions.

1. Quick cursor movements. ‹HOME› allows you to page forward or backward through a file or jump to the end or beginning of a line, depending on the previous cursor movement command.
   - If ‹DOWN› was the previous command, the next page of text is displayed, leaving the cursor in the same column as at the start of the command. (You need to press ‹DOWN› only once for multiple paging; e.g., ‹DOWN› ‹HOME› ‹HOME› ‹HOME› pages three times.)
   - If ‹UP› was the previous command, the previous page of text is displayed, leaving the cursor in the same line as at the start of the command. (You need to press ‹UP› only once for multiple paging.)
   - If ‹LEFT› was the previous command, the cursor moves to the beginning of the line
   - If ‹RIGHT› was the previous command, the cursor moves to the end of the line.
2. Line edit prompts to enter reedit mode.

Count. Count is significant only for < UP > and < DOWN > < HOME > s, where it multiplies the distance moved.

Related Commands. < LEFT >, < RIGHT >, < UP >, < DOWN >

3.2 CARRIAGE RETURN Command

The < RETURN > key moves the cursor to the beginning of the next line. Special cases of < RETURN > are—

- If the cursor is on the last line of the file, < RETURN > moves the cursor to the end of the file.
- If the auto indent option (SET Indent) is on, < RETURN > moves the cursor to the first nonblank character of the next line.
- If the cursor is on the last line of text on the screen, the screen scrolls up one line.

Count. This command accepts any count where count multiplies the distance moved. If count is / or greater than the number of lines that exist in the file from the current cursor position to the end of the file, the cursor jumps to the end of the file.

Related Commands. SET Indent

3.3 < CTRL-C > Function Key

< CTRL-C > serves as a *soft command abort.* A < CTRL-C > is recognized as soon as it is typed, even if a command is in progress. Therefore, < CTRL-C > may be used to terminate a time-consuming command while it is executing, e.g., a long FIND or repeated long macros. The termination is carried out at the first opportunity that allows termination without affecting AEDIT-86's integrity. < CTRL-C > does not abort the execution of an I/O related command. This option is the only way to abort an infinite loop in macro execution gracefully.

3.4 TAB Function Key

Pressing the TAB key causes the next prompt line to be displayed. The prompt line display is cyclic (i.e., after the last prompt line is displayed, the first is redisplayed).

3.5 Delete Function Keys

< RUBOUT > is a specifically labeled key. The other delete function keys are configurable: delete character < DELCH >, delete left < DELL >, delete right < DELR >, and delete line < DELLI >.

3.5.1 < RUBOUT >

The RUBOUT key deletes the preceding character. If a line feed is deleted, the preceding carriage return is deleted also.

There is no recovery from this deletion.

Count. < RUBOUT > ignores count.

3.5.2 <DELCH>

The DELETE CHARACTER command is configurable, usually as <CTRL-F>. It deletes the character "under" the cursor. If a carriage return is deleted, the following line feed (if any) is also deleted because in this case, AEDIT-86 considers the carriage return/line feed pair as a single character.

There is no recovery from this deletion.

Count. This command limits count to 32 to prevent accidental destruction of the file. If count is greater than 32, the message "cannot delete more than 32" is displayed on the message line.

3.5.3 <DELL>

The DELETE LEFT command is configurable, usually as <CTRL-X>. It deletes all characters to the left of the cursor on the line on which the cursor is positioned.

The deletion can be recovered with the <UNDO> command.

Count. The DELETE LEFT command ignores count.

Related Commands. <UNDO>

3.5.4 <DELR>

The DELETE RIGHT command is configurable usually as <CTRL-A>. It deletes all characters to the right of the cursor on the line on which the cursor is positioned, except for the carriage return/line feed pair.

The deletion can be recovered with the <UNDO> command.

Count. The DELETE RIGHT command ignores count.

Related Commands. <UNDO>

3.5.5 <DELLI>

The DELETE LINE command is configurable, usually as <CTRL-Z>. It deletes the entire line on which the cursor is positioned. All lines below the deleted line move up one row. The cursor is left in the same position on the new line.

The deletion can be recovered with the <UNDO> command.

Count. The DELETE LINE command ignores count.

Related Commands. <UNDO>

3.6 <UNDO> Command

The UNDO command is configurable, usually as <CTRL-U>. It restores characters deleted by the last DELETE LEFT, DELETE RIGHT, or DELETE LINE command at the current cursor position. If the previous command was DELETE LINE, the cursor moves to the beginning of the current line before the restoration. Consecutive <CTRL-U>s repeat the restoration of the same string.

Count. The ‹UNDO› command ignores count.

Related Commands. ‹DELL›, ‹DELR›, ‹DELLI›

3.7 ‹HEX› Function Key

The ‹HEX› prefix (usually configured to ‹CTRL-H›) is used to insert a character into the text as its ASCII value. This feature is useful in the following cases:

- When the character to be inserted has no key; e.g., a character with a hexadecimal value over 7FH.
- When the character to be inserted has a special function in the current mode, e.g., if you want to insert the ‹ESC› character into your text.

To insert a character using ‹HEX›, you should type ‹HEX›, then two digits that are interpreted as the ASCII value of the character. ‹HEX› is unseen and not inserted; however, it directs the treatment of the next two characters. For example, if, in INSERT mode, you type ‹HEX›5A, the letter Z will be inserted at the cursor position.

‹HEX› can be activated under INSERT, XCHANGE, and line-edited prompts. It cannot be activated at main command level.

3.8 INSERT Mode

INSERT mode allows you to enter text. To enter INSERT, press I. To exit INSERT mode and return to main command level, press ‹ESC›.

3.8.1 Description

Press I; AEDIT-86 prompts—

```
[insert]
```

The prompt [insert] is displayed whenever AEDIT-86 is in INSERT mode. Move the cursor to any location in the text and begin typing; the characters are inserted into the text.

‹ESC› causes the editor to leave INSERT mode and return to main command level.

The cursor movement commands (‹LEFT›, ‹RIGHT›, ‹UP›, ‹DOWN›, or ‹HOME›), the delete keys (‹RUBOUT›, ‹DELCH›, ‹DELLI›, ‹DELL›, ‹DELR›), and ‹UNDO› all function the same as they do at main command level.

The cursor movement commands and the delete keys restart INSERT mode at the new location.

‹RETURN› inserts a carriage return/line feed pair and moves the cursor to the beginning of the next line.

<CTRL-C> deletes all text inserted since the beginning of INSERT mode, or since INSERT mode was restarted by one of the following commands: cursor movement commands (<LEFT>, <RIGHT>, <UP>, <DOWN>) or delete keys (<DELCH>, <DELLI>, <DELL>, <DELR>), but does not restore characters deleted with <RUBOUT>, <DELCH>, <DELLI>, <DELL>, or <DELR>. After restoration, <CTRL-C> returns the editor to main command level.

The <HEX>, <MEXEC>, <FETN>, and <FETS> functions described here are the same in both INSERT and XCHANGE modes.

<HEX> is unseen and not inserted. The next two characters, which must be hexadecimal digits, are interpreted as the hexadecimal values of a character. <HEX> is described above.

<MEXEC> is the macro execution key. This character is unseen and not inserted. When <MEXEC> is pressed, the next character is treated as a macro name. <MEXEC> is described in detail in Chapter 5.

<FETN> is the fetch numeric key. This character is unseen and not inserted. When <FETN> is pressed, the next character (which must be a numeric value from 0–9) fetches the assigned value. <FETN> is described in Chapter 6.

<FETS> is the fetch string key. This character is unseen and not inserted. When <FETS> is pressed the next character (which must be a numeric value from 0–9 or the second letter from the read-only string variables) fetches the assigned value. <FETS> is described in Chapter 6.

In INSERT mode, macro execution usually restarts the insert process. The only exceptions are non-modeless macros that consist of nonrestarting commands only. (See Section 5.3, "Macro Modes," for a description of modeless and non-modeless macros.)

INSERT mode is modified if it is preceded by a forward slash (/). All text past the cursor in the current line is moved down one line. The text is restored before any delete or move subcommand (except <RUBOUT>) or when insertion is complete.

Count. Repeat count is not a valid option in INSERT mode.

Related Commands. SET Indent, SET Autonl

3.9 XCHANGE Mode

XCHANGE mode allows you to type over text. To enter XCHANGE, press X. To exit XCHANGE mode and return to main command level, press <ESC>.

3.9.1 Description

Press X; AEDIT-86 prompts—

```
[exchange]
```

The prompt [exchange] is displayed whenever AEDIT-86 is in XCHANGE mode. Move the cursor to any location in the text and begin typing; characters are replaced on a one-for-one basis. The carriage return/line feed pair is not replaced; instead, the line is extended.

<ESC> causes the editor to leave XCHANGE mode and return to main command level.

The cursor movement commands <LEFT>, <RIGHT>, <UP>, <DOWN>, <HOME>, and the delete keys (except for <RUBOUT>) <DELCH>, <DELL>, <DELR> and <DELLI>, and <UNDO> work as at main command level.

The cursor movement commands and the delete keys restart XCHANGE mode at the new location.

<RUBOUT> replaces the character to the left of the cursor with the original character (if it has been exchanged), with the following exceptions:

- If the line has been extended, <RUBOUT> works as at main command level and deletes the character immediately to the left of the cursor.
- If the cursor is at the original replacement location, <RUBOUT> moves one character to the left but does not delete the character.

<RETURN> replaces the character "under" the current cursor position with a carriage return/line feed pair and moves the cursor to the beginning of the next line.

<CTRL-C> restores original text (text before it was exchanged). However, once you have exchanged text and pressed <ESC> or restarted XCHANGE with any of the cursor movement commands (<LEFT>, <RIGHT>, <UP>, <DOWN>, or <HOME>), changes cannot be revoked by pressing <CTRL-C>. After restoration, <CTRL-C> returns the editor to the main command level.

The <HEX>, <MEXEC>, <FETN>, and <FETS> functions described here are the same in both INSERT and XCHANGE modes.

<HEX> is unseen and not inserted. The next two characters, which must be hexadecimal digits, are interpreted as the hexadecimal values of a character. <HEX> is described above.

<MEXEC> is the macro execution key. This character is unseen and not inserted. When <MEXEC> is pressed the next character is treated as a macro name. <MEXEC> is described in Chapter 5.

<FETN> is the fetch numeric key. This character is unseen and not inserted. When <FETN> is pressed, the next character (which is a numeric value from 0–9) fetches the assigned value. <FETN> is described in Chapter 6.

<FETS> is the fetch string key. This character is unseen and not inserted. When <FETS> is pressed, the next character (which is a numeric value from 0–9 or the second letter from the read-only string variables) fetches the assigned value. <FETS> is described in Chapter 6.

Macro execution in XCHANGE mode usually restarts the XCHANGE process. The only exceptions are non-modeless macros that contain non-restarting commands only. (See Section 5.3, "Macro Modes," for a description of modeless and non-modeless macros.)

Count. Repeat count is not a valid option in XCHANGE mode.

Error. "Xchange limit is 100" is displayed if you attempt to exchange over 100 characters without restarting XCHANGE mode. XCHANGE mode has a limit of 100 characters.

3.10 FIND Command

The FIND command searches forward from the current cursor position to the end of the file for a string of characters and positions the cursor after the next occurrence of the string.

3.10.1 Description

Press F; AEDIT-86 prompts—

```
---- <HOME> to re-edit
Find {mode} "target_string"
```

The last *target string* (if any) is displayed within quotes. *mode* refers to the SET options currently in effect that may influence the FIND command.

Note that pressing < RETURN > when specifying a target string inserts a carriage return/line feed into the target string and adds the RETURN symbol, < nl >, to the prompt line. You must press < ESC > to complete the string specification and execute the FIND command.

The cursor is placed immediately after the next occurrence of the target string. If the string is not found, the message "not found: "*target string*" " is displayed in the message line and the FIND command is marked as failed.

The following attributes affect operation of the FIND command:

- Case — consider case of Find target
- Showfind — list lines or multiple finds
- K_token — find only token strings

Case, Showfind, and K_token refer to features set using the SET command, which is described later in this chapter.

Case refers to upper- or lowercase letters. If Case is *no*, upper- and lowercase letters are equivalent.

If Showfind is *yes*, each line that contains the target string is displayed.

The Token is the argument between delimiters. Token characters are all characters that are not delimiters. For example, if you want to find the word *is* in the preceding paragraph, but not the *is* in *displayed*. If SET K_token is yes, each occurrence of the word *is* is found because it is enclosed in delimiters.

The string {*mode*} contains an abbreviation for the features, Case (Cs), Token (Tk), and Showfind (Sh). The abbreviation is displayed only if the specified feature is set to yes. If more than one feature is yes, the features are separated by a blank.

The following are examples of mode:

FIND { Cs } "..."	Consider case = yes
FIND { Tk Sh } "..."	Token and Showfind = yes
FIND { } "..."	No option is set

To abort a (-)FIND command, press < CTRL-C >.

The message "found: *count*" is displayed when the command is complete.

Count. The FIND command accepts any count where count indicates the number of times to search for a target string. The search stops after the last occurrence of the target string is found or count is exhausted.

Error. "not found: *target_string*" is displayed if no match is found, and the editor returns to main command level.

Related Commands. SET Case, SET K_token, SET Showfind

3.11 -FIND Command

The -FIND command is identical to the FIND command with the following exceptions:

- -FIND searches backward from the current cursor position to the beginning of the file.
- The Showfind option is ignored.
- The cursor is positioned on the first character of the matched string.

3.11.1 Description

Press -; AEDIT-86 prompts—

```
---- <HOME> to re-edit
-find {mode} "target_string"
```

The last target string (if any) is displayed in the quotation marks.

Related Commands. SET Case, SET K_token

3.12 REPLACE Command

The REPLACE command is similar to the FIND command except that it allows you to replace the old target string with a new string. The REPLACE command also allows you to delete a target string.

3.12.1 Description

Press R; AEDIT-86 prompts—

```
---- <HOME> to re-edit
Replace {mode} "target_string"
```

The last *target string* (if any) is displayed within the quotation marks.

The prompt line contains two line-edited argument. You may edit either or both of them in the following ways:

- Press < HOME > to enter the reedit mode; to enter a new target string simply type the new string.
- Press < ESC > to terminate editing the target string and start editing the replacement string; AEDIT-86 prompts—

  ```
  ---- <HOME> to re-edit
  Replace {mode} "target_string" with "replacement_string"
  ```

- If you want to return to editing the target string, press <RUBOUT> with the cursor on the first position of the replacement string.
- Press <ESC> if you are on the target string and <RUBOUT> if you are on the first character of the replacement string to reedit the target string and the replacement string as necessary.
- Press <ESC> while editing the replacement string to terminate editing the target/replacement strings and to start the replacement process.

To delete a target string from text, the replacement string must be an empty string. The first time you request REPLACE, the replacement string will be an empty string; therefore, press <ESC> to enter an empty replacement string and execute the REPLACE command. If a replacement string from a previous command is displayed, type a character and press <RUBOUT> to remove the character. Then press <ESC> a second time to enter the empty replacement string and execute the command.

Operation of the REPLACE command is affected by the following attributes (equivalent to the FIND command):

- Case — consider case of Find target
- Showfind — list lines or multiple finds
- K_token — find only token strings

Case, Showfind, and K_token refer to features set using the SET command, which is described later in this chapter.

Case refers to upper- or lowercase letters. If Case is *no*, upper- and lowercase letters are equivalent.

If Showfind is *yes*, each line that contains the target string is displayed.

The string {*mode*} contains an abbreviation for the features, Case (Cs), Token (Tk), and Showfind (Sh). The abbreviation is displayed only if the specified feature is set to yes. If more than one feature is yes, the features are separated with a blank.

To abort a REPLACE command, press <CTRL-C>.

The message "found: *count* replaced: *count*" is displayed when the command is complete.

The (?)REPLACE and (-)FIND commands share the same target string and each changes the other's default target.

Count. The REPLACE command accepts any count where count indicates the number of times to replace a target string. Replacement stops after replacing the last occurrence of the target string or count is exhausted.

Error. "not found: *target_string*" is displayed if no match is found, and the editor returns to main command level.

Related Commands. SET Case, SET K_token, SET Showfind

3.13 ?REPLACE Command

The ?REPLACE command is the conditional REPLACE command.

3.13.1 Description

Press ?; AEDIT-86 prompts—

```
---- <HOME> to re-edit
?Replace {mode} "target_string"
```

The ?REPLACE works exactly the same as the REPLACE command except that the following prompt is displayed on each find:

```
ok to replace?  (y or [n])
```

If *y* (or *Y*) is typed, the replacement is made. Any other key is considered a negative response.

3.14 TAG Command

The TAG command allows you to specify four locations in a file and, with the JUMP command, move the cursor to one of these locations. The TAG command relates to the current cursor position. Tags are invisible and are not saved when you exit the file. After the tag is set, the editor automatically returns to main command level.

3.14.1 Description

You can set four tags: A, B, C, and D.

The cursor's current position determines the tag location.

Press T; AEDIT-86 prompts—

```
A_tag   B_tag   C_tag   D_tag
```

The tag is set by pressing A, B, C, or D.

If the section containing the tag is deleted, the tag is "moved" to the first position after the deleted section.

Each input file (main or OTHER) has its own set of tags.

Count. The TAG command ignores count.

Error. "invalid command" is displayed if a key other than A, B, C, or D is pressed. AEDIT-86 returns to main command level.

Related Commands. JUMP

3.15 JUMP Command

The JUMP command moves the cursor to a specified location in text. The editor automatically returns to main command level.

3.15.1 Description

Press J; AEDIT-86 prompts—

```
A_tag   B_tag   C_tag   D_tag   Start   End   Line   Position
```

Count. The JUMP command ignores count.

A_tag, B_tag, C_tag, D_tag

The Tag subcommand is executed by pressing A, B, C, or D. The cursor jumps to the specified tag, previously set with the TAG command.

Error. "no such tag" is displayed if the specified tag does not exist.

Related Command. TAG

Start

The Start subcommand is executed by pressing S. It moves the cursor to the start of the file.

End

The End subcommand is executed by pressing E. It moves the cursor to the end of the file.

Line

The Line subcommand is executed by pressing L. It prompts for a decimal value:

```
line:
```

The cursor jumps to the start of the designated line. (The first line of a file is number 1; the maximum value allowed is 65536.) If the value entered is greater than the number of lines in the file, the cursor jumps to the EOF marker.

Error. "illegal value" is displayed if a value larger than the maximum value or any other illegal value is entered.

Position

The Position subcommand is executed by pressing P. It prompts for a decimal value:

```
column:
```

The cursor jumps to the designated position in the current line. The first position in a line is 0; the maximum value allowed is 254.

- If the current line is shorter than the designated position value, the cursor jumps to the last position in the line.
- If there is no character at the specified position (due to tab expansion), the cursor jumps to the next character.

Error. "illegal value" is displayed if a value larger than the maximum value or any other illegal value is entered.

3.16 BLOCK Command

The BLOCK command is invoked by pressing B (or D; typing either B or D initially is equivalent). The BLOCK command allows you to delimit a section of text that can then be deleted, moved, or copied. The subcommand determines if the text will be deleted, moved, or copied. The Buffer subcommand copies the delimited section to the Block buffer. The Delete subcommand deletes the delimited section and places it in the Block buffer. The Put subcommand copies the section to an external file.

The commands (-)FIND, JUMP, and the cursor movement commands < LEFT >, < RIGHT >, < UP >, < DOWN >, < HOME >, and < RETURN > work the same as at main command level.

Text saved in the Block buffer (or in an external file) can be retrieved with the GET command. The GET command copies the contents of the Block buffer (or external file) at the current cursor position in your file.

Pressing < ESC > returns the editor to main command level with the cursor left in its current position.

Pressing < CTRL-C > returns the editor to main command level with the cursor at its original position at the beginning of the delimited section.

3.16.1 Block Buffer

The Block buffer is the buffer associated with the BLOCK and DELETE (and GET) commands. It has a fixed maximum size of 2 Kbytes. If the section to be copied to the Block buffer is over 2 Kbytes, the remainder is written to a temporary work file. The contents of the buffer remain unchanged until you execute another BLOCK or DELETE command, when a new section of text is written to the buffer, overwriting the old contents.

3.16.2 Description

To delimit a section of text, move the cursor to the first character of the section. Press B or D; the @ sign serves as the delimiter and replaces the character marking one endpoint of the block. Endpoints can be set in either order; this example sets the beginning endpoint first.

Press B (or D); AEDIT-86 prompts—

```
Buffer  Delete  Find  -find  Jump  Put
```

Move the cursor to the character immediately after the block to be delimited. The character under the first @ sign is included in the block, but the character under the second @ is not.

Now specify one of the subcommands by pressing the initial letter of that subcommand.

Count. The BLOCK command ignores count.

Related Command. GET

Buffer

To execute the Buffer subcommand, press B. It copies text to the Block buffer. The @ signs are removed; the delimited section of text is copied to the Block buffer. The delimited section of text is not affected.

Delete

To execute the Delete subcommand, press D. It deletes the delimited section from the text and moves it to the Block buffer. If the deleted text does not fit in the portion of the Block buffer that is in memory, the menu prompts—

```
cannot save in memory--save anyway?  ([y] or n)
```

If *n* is specified, the delimited section of text is deleted, but the Block buffer is not updated. If you press any other key, the delimited section is written to a temporary file. Press <CTRL-C> to abort the command.

Find

To execute the Find subcommand, press F. It works the same as it does at main command level.

-find

To execute the -find subcommand, press the hyphen (-). It works the same as it does at main command level.

Jump

To execute the Jump subcommand, press J. It works the same as it does at main command level.

Put

The Put subcommand allows you to copy a section of text to a named output file.

Press P; AEDIT-86 prompts—

```
---- <HOME> to re-edit
Output file:  filename
```

The filename of the previous command (if any) is displayed to the right of the colon. The delimited section of text may be copied to this file or a different filename may be specified. The delimited section is not affected.

If the specified file already exists, the editor beeps and the following prompt is displayed:

```
overwrite existing file?  (y or [n])
```

If *y* is specified, the delimited text is written to the named file, overwriting the previous file.

If any other key is typed, the editor returns to the BLOCK Put prompt level.

The specified file can also be written to an output device (e.g., :lp:) supported by your system.

3.17 DELETE Command

To invoke the DELETE command, press D. The DELETE command exists to enable deletion of a section of text by typing D at both endpoints.

3.18 GET Command

The GET command retrieves the contents of the Block buffer or an external file and inserts it at the current cursor position in your file.

3.18.1 Description

Move the cursor to the point in your file where you want the contents of the buffer (or external file) to be placed.

Press G; AEDIT-86 prompts—

```
---- <HOME> to re-edit
Input file:  filename
```

The filename previously specified for the Get command (if any) is displayed to the right of the colon.

To insert the contents of the Block buffer at the current cursor location, press < RETURN >. To insert the contents of an external file, type the name of the file, then press < ESC > or < RETURN >.

The editor returns to main command level with the cursor on the first inserted character.

Count. This command accepts any finite count. The named file is copied to the current location *count* times. The repeat count / is not valid with the GET command. If /G is typed, AEDIT-86 returns to main command level without issuing an error message.

Related Command. BLOCK

3.19 VIEW Command

To execute the VIEW command, press V. This command allows you to rewrite (move) the text on the screen so that the row containing the cursor (the viewrow) is positioned on the row that you have defined. In this way you may rewrite (move) the text area you are editing to position the viewrow on line 10, for example. VIEW is also useful to refresh the screen image if it has been corrupted (e.g., power failure, terminal problems). The viewrow is set with the SET Viewrow command; the default is 5. Reverse scroll is implemented as VIEW on terminals without reverse scroll.

VIEW also issues an abbreviated sign-on message, which includes the busy/waiting indicator, system-id, and AEDIT-86 version number.

Count. The VIEW command ignores count.

Related Command. SET Viewrow, WINDOW

3.20 OTHER Command

AEDIT-86 has two distinct buffers. The text area used at startup is called the main buffer; the other is called the OTHER buffer. To execute the OTHER command, press O. It allows you to switch from editing text in one buffer, the current buffer, to editing text in the other or secondary buffer.

Pressing O a second time returns the editor to the main buffer.

Main and OTHER buffer text may be displayed simultaneously by splitting the screen into two windows using the WINDOW command.

3.20.1 Description

Press O; the message "Other Editing *input file*" or, if no input file has been specified, "Other no input file" is displayed at the start of the message line whenever the secondary text is displayed:

```
---- Other Editing input file
Again  Block  Calc  Delete  Execute  Find  -find  --more--
```

The OTHER buffer allows you to edit a second file in the same way as the main buffer.

It is possible to display both the main and the OTHER text buffers by splitting the screen into two windows using the WINDOW command, then typing O to display the OTHER file in one window. In this case, switching from one window to the other results in switching from one text buffer to the other.

Each partition has a separate set of tags. A JUMP command is valid only within its own partition; it cannot jump to the other partition. Also, each partition has its own value for SET Leftcol. The Block buffer is common to allow moving text between partitions.

Press O to exit the OTHER buffer and return to the main buffer.

Count. Count has no meaning for the OTHER command.

3.21 AGAIN Command

To execute the AGAIN command, press A. It causes the last command or in some cases the last subcommand to be repeated.

In the following commands, AGAIN repeats the entire command, including subcommand arguments:

- PARAGRAPH
- (-)FIND, (?)REPLACE

In the following command, AGAIN repeats the entire command including subcommands, but without their arguments:

- HEX command

Count. Count is the count for the repeated command. The value of count given for the last command is ignored.

3.22 SET Command

The SET command allows you to set/reset several features that determine how AEDIT-86 will operate, e.g., if case (uppercase, lowercase) should be considered in the target string of a search command.

Most SET subcommands relate to switches. A switch is an option that has only two states: yes or no. When a subcommand of this type is activated, a yes/no question is displayed on the prompt line. The value currently in effect is enclosed in square brackets. Each feature has a default value; this value is in effect until it is reset using the SET command.

3.22.1 Description

Press S (press ‹TAB› to view the remaining prompt lines); AEDIT-86 prompts—

```
Autonl Bak_file Case    Display E_delimit Go     Highbit  --more--
Indent K_token  Leftcol Margin  Notab     Radix  Showfind --more--
Tabs   Viewrow                                            --more--
```

To specify an option, press the initial (uppercase) letter of that option.

Count. The SET command ignores count.

Autonl

To activate the Autonl option, press A. This option allows you to automatically create a new line at the right margin, in INSERT mode. It prompts—

```
insert cr,lf automatically? (y or [n])
```

- If *y*, carriage return/line feed (new line) is inserted in the last position on the screen whenever an attempt is made to insert a character in that position. If the character to be inserted is not a white space (i.e., not a space, tab, line feed, or carriage return), the carriage return/line feed pair is inserted before the token, if possible. Trailing blanks and tabs are deleted, and the carriage return/line feed is inserted between words. The last position is set using SET Margin (described later).
- If *n*, the option is turned off.

Related Commands. INSERT, SET Margin

Bak_file

To execute the Bak-file option, press B. It saves a backup file that contains the last version of your file. It prompts—

```
create .BAK files? ([y] or n)
```

- If *y*, the file you are editing is renamed file.BAK when QUIT Exit or QUIT Update is executed to save the revised version of the file.
- If *n*, this option is turned off.

NOTE

Turning off this option can be dangerous. If your file is accidentally lost or damaged and Bak-file is yes, the most recent version of the file has been saved in the backup file.

Case

To execute the Case option, press C. It considers the case of the target string in a search command. It prompts—

```
consider case of Find target? (y or [n])
```

- If *y*, you can type the target string in uppercase, lowercase, or a combination of both, and the case is significant.
- If *n*, you can type the target string in uppercase, lowercase, or a combination of both, but the case is ignored.

For example,

If the target string for the FIND command is *tHe* and the option is set to y, AEDIT-86 will find *tHe* only.

If the target string for the FIND command is *tHe* and the option is set to n, AEDIT-86 will find *tHe*, *the*, *THE*, etc.

Related commands. (-)FIND, (?)REPLACE

Display

To execute the Display option, press D. This options allows you to display the text changes resulting from macro execution. It prompts—

```
display macro execution?  (y or [n])
```

- If *y*, during macro execution all cursor movements and text changes are displayed on the screen.
- If *n*, this option is turned off. Thus, when macro execution starts, cursor movements or changes in the text are not displayed on the screen.

For example, if you execute a macro and SET Display Yes is in effect, sections of text that are changed by the macro execution are displayed on the screen each time the macro is executed.

E_delimit

To execute the E_delimit option, press E. The current delimiter set may be displayed and new delimiters set using this option. SET E_delimit is used by the FIND/REPLACE commands under token mode. It prompts—

```
---- <HOME> to re-edit
delimiter set:  !"#%&'()*+,-./:;<=>?@[\]^`{|}~
```

All characters currently specified as delimiters are displayed to the right of the colon on the prompt line.

Delimiters have the following properties:

- A delimiter is always one character.
- Characters with hexadecimal values from 00H–20H, and 7FH or more are predefined delimiters. They are not displayed, and they cannot be excluded from the delimiter set.
- ASCII characters with the values 21H–7EH are displayed (if specified).
- Delimiters are displayed with no separating characters

Delimiters specified by SET E_delimit are used to define a token for the (-)FIND/(?)REPLACE operation. A token is any nonempty string surrounded by delimiters.

When you specify a set of delimiters, you may include the same delimiter more than once. For example, you may separate input delimiters with blanks.

The default E_delimit string is—

```
!"#%&'()*+,-./:;<=>?@[\]^`{|}~
```

Related Commands. (-)FIND, (?)REPLACE, SET K_token

Go

To execute the Go option, press G. It relates to macro execution continuation after a (-)FIND/(?)REPLACE command failed. It prompts—

```
continue macro execution after a failure?  (y or [n])
```

This option is meaningful only in macro execution; it has no meaning at main command level. The default for the SET Go option, when a macro is started, is No for all macros regardless of the current setting of the option. To use this option, you should specify SET Go Yes in your macro. It may be reset within the same macro file. The SET Go option does not affect a nested macro.

- If *y* is in effect for the current macro file and a (-)FIND/(?)REPLACE command fails, execution of the current macro continues, i.e., the next command is activated.
- If *n* is in effect for the current macro file, and a (-)FIND/(?)REPLACE command fails, execution of the current macro is terminated, and control is returned to the caller, either a macro or main command level.

During MACRO Create, the SET Go command is inserted into the macro definition, but the macro currently defined is executed as if SET Go is Yes.

Related Commands. MACRO Create, EXECUTE

Highbit

To execute the Highbit option, press H. It allows the display of characters with hexadecimal values over 7FH. It prompts—

```
display parity-on characters as is?  (y or [n])
```

- If *y*, all text characters with hexadecimal values over 7FH are written to the screen as is.
- If *n*, all text characters with hexadecimal values over 7FH are displayed as ?.

Indent

To execute the Indent option, press I. This option is useful when entering code for a structured language such as PL/M or Pascal. It prompts—

```
automatically indent during insertion?  (y or [n])
```

- If *y* in INSERT mode and < RETURN > is pressed, the next line is automatically indented to the position of the first character of the preceding line. For the first line (after a blank line), pressing < RETURN > moves the cursor to position 0 of the next line. At main command level, the cursor moves to the first nonblank, nontab character in the next line.
- If *n*, this option is turned off.

The Indent option is not active if a carriage return is inserted using the < HEX > prefix (< HEX > 0D) or if a line feed is inserted without a carriage return.

This option is not active in XCHANGE mode.

Related Commands. Carriage Return, INSERT

K_token

To execute the K_token option, press K. This option allows you to find a string only if it is enclosed by delimiters and is not part of a larger string. It prompts—

```
find only token strings?  (y or [n])
```

Token characters are all the characters that are not delimiters. A token is defined as a nonempty string surrounded by delimiters. Delimiters in this context are the characters specified in SET E_delimit.

- If *y*, a string is found by the (-)FIND or (?)REPLACE command only if the string fits the token definition.

  A string in the text that is found by (-)FIND or (?)REPLACE commands when SET K_token No is in effect will also be found when SET K_token Yes is in effect ONLY if that string is a nonempty string surrounded by delimiters. Delimiters include the beginning and end of a file, the cursor position, carriage return and line feed.
- If *n*, a string in the text is found regardless of the characters that surround it.

Related Commands. (-)FIND, (?)REPLACE, SET E_delimit

Leftcol

To execute the Leftcol option, press L. The option allows viewing lines with more than 80 characters. It prompts—

```
---- <HOME> to re-edit
left column: 0
```

The current left column is displayed to the right of the colon. This command allows you to view lines over 80 characters long on the screen and accepts any number from 0 to 175 (position count starts at 0). The number input indicates the number of characters at the start of a line that should not be displayed.

For example, if a line is 90 characters long, you can set Leftcol to 20 and the screen will display the line from position 21 to the end of the line.

An exclamation point (!) is printed in column 0 when characters to the left are not displayed.

Leftcol can also be set by typing the plus sign (+) or the minus sign (−) followed by a valid decimal number. This sets the left column at the current value plus or minus the number given.

AEDIT-86 may have two different values for Lefcol simultaneously, one for the main file and one for the OTHER file. The default Leftcol for both files is zero.

For example, if the left column is currently set at position 15 and you type S(ET) L(eftcol) -10, the new left column is position 5.

Error. "bad Leftcol" is displayed if you attempt to set a value out of range.

Margin

To execute the Margin option, press M. It allows setting values for indenting, and for left and right margins for reformatting a paragraph. It prompts—

```
---- <HOME> to re-edit
indent, left, right:  4 , 0 , 76
```

The current values for indent, left, and right are displayed to the right of the colon separated by commas. The first number sets the indentation, the second the left margin, and the third the right margin. Indent may be set at any value from 0–253; left, from 0–253; and right, from 1–254. The value of the right margin must be greater than the indentation and the left margin. When entering the values, separate them by one or more blanks or a comma. The values of all three numbers are absolute and offset from position 0. The default values are 4, 0, and 76.

Press **<** ESC **>** or **<** RETURN **>** to execute the command and return to main command level.

To set indent to 0, left to 5, and right to 70, type—

S(ET) M(argin) `0, 5, 70 <RETURN>`

To reset the left margin to 2, type—

S(ET) M(argin) `, 2 <RETURN>`

Related Commands. PARAGRAPH, SET Autonl

Notab

To execute the Notab option, press N. This option instructs the editor to replace inserted tabs with the appropriate number of blanks. It prompts—

```
insert blanks for tabs? (y or [n])
```

- If *y*, blanks are inserted instead of tabs whenever you press the **<** TAB **>** key in the INSERT or XCHANGE mode.
- If *n*, this option is turned off.

Note that this option does not affect tabs that are entered using the **<** HEX **>** prefix (**<** HEX **>** 09).

Related Commands. SET Tab

Radix

To execute the Radix option, press R. This option allows you to determine the radix (base) in which an AEDIT-86 variable will be inserted in the text. It prompts—

```
---- current Radix: radix
Alpha  Binary  Decimal  Hex  Octal
```

This option affects values inserted or exchanged by the Fetch operation **<** FETN **>** (described in Chapter 6). The Radix default is Decimal.

For more on Radix, see Chapter 6.

Related Commands. INSERT, XCHANGE

Showfind

To execute the Showfind option, press S. It is used to display all lines containing the target string in a FIND/REPLACE command. It prompts—

```
list lines on multiple finds? (y or [n])
```

- If *y*, when you execute a FIND or (?)REPLACE command, the screen is cleared, and each text line that contains the target string is displayed on the screen. Each string is displayed on a separate line. When the screen is full, the message "hit space to continue" is displayed.
- If *n*, the FIND and (?)REPLACE commands execute as usual, but the screen is not cleared, and text lines that contain the string are not displayed on the screen.

Related Commands. FIND, (?)REPLACE

Tabs

To execute the Tabs option, press T. This option allows you to set tabs. It prompts—

```
---- <HOME> to re-edit
Tabs: 4
```

The prompt line lists the current tab settings. If you want only to inspect the tab settings, type < CTRL-C > to return to main command level.

To enter tabs, type a list of decimal numbers separated by at least one blank or a comma. The numbers must be specified in increasing order, from 1–253. Changing the tab settings does not change the file contents, but it may affect its display on the screen.

The default tab settings are every fourth position, i.e., 4, 8, 12... .

The difference between the last two numbers specified for tabs is repeated, up to 253.

For example—

4	sets tabs at 4, 8, 12, 16, ...
5,6,10	sets tabs at 5, 6, 10, 14, 18, ...
0	sets tabs at 4, 8, 12, 16, ...

NOTE

Columns start at 0, not 1. Therefore, FORTRAN tabs should be 6,10, not 7,11.

Errors. "bad tabs" is displayed if you attempt to set an illegal tab.

Related Commands. INSERT, XCHANGE

Viewrow

To execute the Viewrow option, press V. This option allows you to select a viewrow for rewriting the screen. It prompts—

```
row for View:  5
```

The current viewrow setting is displayed to the right of the colon. Type the number of the row on which you wish the cursor to be positioned by the VIEW command. This value must be between 0 and the text size −1. If, for example, your screen size is 25 rows, then text size is 23 (25 minus the message line and the prompt line). Therefore, in this case, the legal values are 0–22.

The default viewrow is the terminal length divided by 5, which means 5 on most terminals.

If the screen is split using the WINDOW command, the view row is determined separately for each window by a formula based on the current view row and the window size.

Errors. "bad View row" is displayed if viewrow is set to a value greater than text size −1.

Related Commands. VIEW, WINDOW

3.23 HEX Command

The HEX command allows you to insert the ASCII equivalents of hexadecimal values in the text. This command also displays the hexadecimal values of text contents in the message line.

3.23.1 Description

Press H; AEDIT-86 prompts—

```
Input  Output
```

To specify a subcommand, press the initial letter of that subcommand.

Input

Press I; AEDIT-86 prompts—

```
---- <HOME> to re-edit
Hex value:
```

The last values entered for HEX input are displayed to the right of the colon. Legal input values consist of one or more strings separated by one or more blanks. A legal string has the following characteristics:

- Every character is a valid hexadecimal digit (0–9, A–F)
- Contains 1 or an even number of characters

Values entered are regarded as hexadecimal; therefore, the suffix H is an illegal character. The values may be separated by one or more blanks.

The following are examples of legal input values:

```
9
5A5B60
3456  65  78F0  8    98C8A7
```

If the input is legal, the equivalent characters are inserted in the text at the cursor position.

Error. "invalid hex value" is displayed if an illegal value is entered, and the editor returns to main command level.

Output

Press O.

The hexadecimal value of the character immediately to the right of the actual cursor position is displayed on the message line. The count that preceded the HEX command gives the number of bytes to be displayed in hexadecimal format. Up to 10 bytes of hexadecimal values can be displayed in the message line. If more bytes are to be displayed, the message "hit space to continue" is displayed.

Press the space bar and the next 10 bytes are displayed. Any other key returns AEDIT-86 to the main command level.

3.23.2 Examples

Example 1

To insert the form-feed character (hexadecimal value 0C) to the current location, type—

H(EX) I(nput) C<RETURN>

Example 2

To insert the digits 1, 2, and 3 to the text, type—

H(EX) I(nput) 31 32 33<RETURN>

Position the cursor over the one and type 3 H(EX) O(utput) to display the characters. The characters are displayed on the message line.

3.24 QUIT Command

The QUIT command performs several functions depending on its subcommand. It ends the editing session, it initializes processing a new input file, it updates your edited file, etc. QUIT has different prompts depending on whether or not a filename has already been specified for the file you are editing.

3.24.1 Filename Specified

Assume you are editing a file with a filename already specified.

Press Q; AEDIT-86 prompts—

```
---- Editing input file [to output file]
Abort    Exit    Init    Update    Write
```

To specify a subcommand, press the initial letter of that subcommand.

Abort

To execute the Abort subcommand, press A. When Abort is activated, if any changes have been made to the current file, AEDIT-86 prompts "all changes lost? (y or [n])"to avoid inadvertent loss of text. A y continues the abort process; the same procedure is then applied to the secondary file. Control returns to the operating system only after both questions have been answered with *y*. All changes (if any) that were made to the input file(s) are lost. If either question is answered with a response other than *y*, AEDIT-86 returns to main command level.

Note that—

- You are questioned concerning only input files that have been changed.
- The first question (if any) relates to the current file and the second one (if any) relates to the secondary file.
- If you answer *y* to the first question and n to the second while AEDIT-86 is still active, the next time you type QUIT Abort, you will not be asked further about the file about which you have already answered *y* unless new changes have been made.

Exit

To execute the Exit subcommand, press E. When activated, AEDIT-86 rewrites the current file. Then, if the secondary file has also been changed, AEDIT-86 automatically performs the OTHER command and asks "all changes lost? (y or [n])". A *y* returns AEDIT-86 to the operating system without rewriting the secondary file. Any response other than *y* returns the editor to main command level.

Init

To execute the Init subcommand, press I. It allows you to start editing a new file without returning to the operating system. If any changes have been made to the current file, the menu prompts "all changes lost? (y or [n])".

Press I; AEDIT-86 prompts—

```
---- <HOME> to re-edit
enter [file [TO file | VO | FO ]]:
```

Enter the input file (or < RETURN >) followed optionally either by an output file name or by the VO or FO controls. File is the file you want to edit, TO file indicates the output file, VO is the abbreviation of the control VIEWONLY; FO is the abbreviation of FORWARDONLY. The TO option can be used only when the input file exists. These controls are described in Chapter 4.

Update

To execute the Update subcommand, press U. It writes the updated version of your file without returning to the operating system.

After the file has been written, the message "*file* has been written" is displayed.

After completing the QUIT Update command, the editor is at the QUIT prompt level, not at main command level.

Write

To execute the Write subcommand, press W.

The following prompt is displayed—

```
---- <HOME> to re-edit
Output file:
```

Enter the output filename. If the specified file exists, the editor beeps and the following question is displayed:

```
overwrite existing file?  (y or [n])
```

- If *y*, the entire text file is written to the named file overwriting the existing file. AEDIT-86 returns to the QUIT prompt level.
- Any other response returns the editor to the QUIT Write prompt level.

After the file has been written, the message "*file* has been written" is displayed.

The QUIT prompt is always reissued after an Update or Write subcommand. Press < ESC > or < CTRL-C > to return to main command level.

Note that:

- QUIT Abort and QUIT Exit relate to the entire AEDIT-86 session, i.e., to both the current file and the secondary file. QUIT Init, QUIT Update, and QUIT Write relate only to the file you are currently editing.
- An output file indicated by the TO clause can be specified for a file in either the invocation line or at the QUIT Init command. Only the subcommands Update and Exit relate to this output file; Write does not.

3.24.2 Filename Not Specified

If you are editing a new file and have not yet specified a filename, the Exit and Update subcommands are not available (both subcommands require a filename). The AEDIT-86 prompt is altered as follows:

```
---- no input file
Abort           Init            Write
```

The subcommands function the same as discussed above.

3.25 PARAGRAPH Command

To invoke the PARAGRAPH command, press P. A paragraph is defined as one or more nonempty lines preceeded by and followed by an empty (or blank) line. The PARAGRAPH command reformats a paragraph using the values set for indent and left and right margin in the SET Margin command. Reformatting means rearranging or adjusting the words within a paragraph between the right and left margins. A word is defined as a sequence of characters surrounded by white spaces (i.e., space, tab, line feed, or carriage return). The default is to reformat one paragraph.

NOTE

The PARAGRAPH command execution starts by indentifying the beginning of the current paragraph. AEDIT-86 searches backward for an empty line; then, the end of the paragraph is found by searching forward for an empty line. This implies that if your file has no empty lines, PARAGRAPH will process the entire file as one paragraph, wherever the cursor is currently positioned in the file.

3.25.1 Description

Move the cursor to any position in the paragraph to be reformatted.

Press P; AEDIT-86 prompts—

```
Fill  Justify
```

To specify an option, press the initial letter of that option.

Count. Count defines the number of consecutive paragraphs to reformat.

Related Commands. SET Margin

Fill

Press F to perform filling. This means that the white space sequences are reduced to one blank, and every blank after a sentence terminator (e.g., period, question mark) is extended to two blanks. Words are moved to fill the line between the right and left margins or from line to line if necessary. The first line is indented according to the value of indent. Words are moved to the left as much as possible. Words are not split and lines are not right-justified.

Justify

Press J to perform justification. The first step is as described in the filling process. The second step is performed separately for each line: words are shifted to the right (if necessary), and the space between words expanded so that the last word of every line ends at the right margin and the spaces between words are approximately even. The last line of the paragraph is not right-justified.

3.26 WINDOW Command

To invoke the WINDOW command, press W. It splits (horizontally) the text area of the screen into two partitions. Each partition or window contains the text, message, and prompt sections. The WINDOW command allows you to view two different parts of the same file or two different files, using the main file and the OTHER file. You may, for example, view errors in a listing file in one window and correct the source file in the other window. The existence or absence of windows does not affect the editing commands. You may think of each window as a narrower screen.

The screen is split above the cursor row. If the cursor is placed so that one window's size is less than five rows, the screen is not split, and the message "window too small" is displayed. After the screen is split, pressing W causes the cursor to jump between the two windows.

At any given time, only one window is active—the window containing the cursor.

When the screen is split using the WINDOW command, the view row is determined separately for each window by a formula based on the current view row and the window size.

If the same file is displayed in both windows, a change in one window is not reflected in the other window until you press W and jump to the other window.

To return to viewing one screen, see the KILL_WND command.

Related Commands. KILL_WND, SET Viewrow, VIEW

3.27 KILL_WND Command

To invoke the KILL_WND command, press K. It returns the screen to one window. The current (active) window is the dominant one. When the KILL_WND command is given, the current window is extended to fill the screen; the cursor remains in its current position. If the file displayed in the current window is at the beginning of the file when the KILL_WND command is given, it scrolls upward to fill the entire screen. This mechanism is indifferent to whether the two windows display two different files or not.

Related Command. WINDOW

3.28 CALC Command

To invoke the CALC command, press C. It provides you with computation capabilities.

3.28.1 Description

Press C; AEDIT-86 prompts—

```
---- <HOME> to re-edit
Calc:
```

The last statement entered under the CALC command is displayed to the right of the colon.

This command is described in detail in Chapter 7.

3.29 EXECUTE Command

To invoke the EXECUTE command, press E. EXECUTE is used to execute Macros.

3.29.1 Description

Press E; AEDIT-86 prompts—

```
---- <HOME> to re-edit
Macro name:
```

The last macro name entered for this command is displayed to the right of the colon.

When typing the macro name is completed, the specified macro is executed.

This command is described in detail in Chapter 5.

Related Command. SET Display, SET Go

3.30 MACRO Command

To invoke the MACRO command, press M. It is used for manipulating macros.

3.30.1 Description

Press M; the menu prompts—

```
Create  Get  Insert  List  Save
```

To specify a subcommand, press the initial letter of that subcommand.

When a macro file is specified in the invocation line (explicitly or implicitly), it is read and processed immediately after AEDIT-86 invocation. This has the same effect as using a MACRO Get as the first command after invocation.

Macro files are described in Chapter 5; AEDIT-86 invocation is described in Chapter 4.

Related Command. SET Go

CHAPTER 4
AEDIT-86 INVOCATION

This chapter describes AEDIT-86 invocation and the AEDIT-86 controls: FORWARDONLY, VIEWONLY, RECOVER, MACRO, MACROSIZE and BATCH.

4.1 Invocation

Invocation details for a particular system are given in the appendixes. Following is the syntax used to invoke AEDIT-86:

```
[device] AEDIT  [input_file  ]  TO  output_file  |  file processing_mode  ]  ]
      [recover]
      [ ,  other_input_file  [  TO  other_output file  |  file processing_mode  ]  ]
      [execution_mode]
```

where

input file	is the file you want to edit. If a file name is not specified, a new file is created. It is named when you call the QUIT command.
output file	is the name of the output file. It is the destination file for the file you are editing if an output file is specified. It is written when you call QUIT Update or QUIT Exit. If either VIEWONLY or FORWARDONLY is specified for the input file, an output file cannot be specified.
other input file *other output file*	supply filenames for the OTHER buffer.
file processing mode	[VIEWONLY | NOVIEWONLY] [FORWARDONLY | NOFORWARDONLY]
recover	[RECOVER | NORECOVER]
execution mode	[MACRO [(*macro_file*)] | NOMACRO] [MACROSIZE (*macro_buffer_size*)] [BATCH | NOBATCH]

A comma is used to separate the main filename from the OTHER filename.

4.1.1 Invocation Line Examples

Example 1

AEDIT-86 can be invoked simply by naming the file you want to edit, as follows:

```
AEDIT MYTEXT.SCR<RETURN>
```

Example 2

This example shows invoking AEDIT-86 with a main input file, an OTHER input file, and a macro file:

```
AEDIT A1.TXT,A2.TXT MACRO(TXT.MAC)<RETURN>
```

Example 3

The following example shows an invocation of AEDIT-86. The main input file and the OTHER input file have the same filename but different filename extensions. The controls MACRO and MACROSIZE are also specified. The main filename and the OTHER filename can be shortened using a hyphen (-), as follows:

```
EDIT PROG.SRC-LST VIEWONLY NOBATCH MACRO(PROG.MAC) MACROSIZE(1024)
```

NOTE

If the TO option is used and the input file does not exist, then the "File does not exist" message is displayed following the invocation line, and AEDIT-86 behaves as if no input or output values were specified.

4.2 Invocation Controls

AEDIT-86 controls can be divided into three groups: file processing mode controls, recover control, and execution mode controls. Table 4-1 lists the AEDIT-86 invocation controls. The remainder of this chapter explains each control in detail.

The processing mode for an input file uses the VIEWONLY and FORWARDONLY controls.

- These controls can be specified for either the main input file, for the OTHER input file, or both. VIEWONLY and FORWARDONLY cannot be specified together for the same file. However, if either of them is specified in the negative form, any combination is legal, e.g., NOVIEWONLY FORWARDONLY.
- A filename must be given if either of these controls is specified in the positive form in the invocation line.

Table 4-1. AEDIT-86 Invocation Controls

Control Name	Abbreviation	Default	Meaning
File Processing Mode Controls			
FORWARDONLY/NOFORWARDONLY	FO/NOFO	NOFO	Enables faster editing of large files, but the size of the edited file is limited.
VIEWONLY/NOVIEWONLY	VO/NOVO	NOVO	Enables fast viewing of large files; no changes allowed.
Recover Control			
RECOVER/NORECOVER	RC/NORC	NORC	Enables file reconstruction.
Execution Mode Controls			
MACRO/NOMACRO	MR/NOMR	MR(*root*.MAC)	Specifies macro file.
MACROSIZE	MS	MS(3072)	Allocates macro buffer size.
BATCH/NOBATCH	BA/NOBA	NOBA	Activates AEDIT-86 in non-interactive mode; used if AEDIT-86 is activated from a command file.

- When an output file is specified, these controls may be specified only in their negative form, e.g., NOVIEWONLY.
- VIEWONLY and FORWARDONLY are the only controls that can be specified under QUIT Init.

The processing mode for the main input file uses the RECOVER control.

- This control may be specified only in the invocation line and only for the main input file.

The execution mode uses the MACRO, MACROSIZE, and BATCH controls.

- These controls may be specified once per AEDIT-86 invocation. They cannot be specified for a particular file, and they cannot be specified under QUIT Init.

A control may be specified only once, except for VIEWONLY and FORWAR-DONLY, which may be specified once for the main input file and once for the OTHER input file.

4.2.1 FORWARDONLY

Syntax

```
FORWARDONLY | NOFORWARDONLY
```

Abbreviation

```
[NO]FO
```

Default

```
NOFORWARDONLY
```

Control Type

Processing mode for an input file

Description

FORWARDONLY allows much faster editing of large files because it instructs AEDIT-86 to allocate a fixed amount of memory for the file. If the file is larger than the amount of memory allocated, some text may be lost. This loss applies only to the current editing of the file; the original file is not affected.

FORWARDONLY can be specified for either the main input file or the OTHER input file. FORWARDONLY can also be specified under the QUIT Init command. FORWARDONLY cannot be specified simultaneously with VIEWONLY. When FORWARDONLY is in effect for the input file, an output file may not be specified.

While the FORWARDONLY control is in effect, the message line displays the word Forward.

Error. "some text lost" is displayed if text is lost during the current edit. If this error is displayed, you are unable to execute QUIT Update or QUIT Exit; however, you may execute QUIT Write.

4.2.2 VIEWONLY

Syntax

```
VIEWONLY | NOVIEWONLY
```

Abbreviations

```
[NO]VO
```

Default

```
NOVIEWONLY
```

Control Type

Processing mode for an input file

Description

A large file, e.g., a large listing file that you do not want to change, may be viewed much faster using the VIEWONLY control. It is also an advantage to use VIEWONLY if you want to be certain that no changes are made unintentionally.

VIEWONLY can be specified for either the main input file or the OTHER input file. VIEWONLY can also be specified under the QUIT Init command. VIEWONLY cannot be specified simultaneously with FORDWARDONLY. When VIEWONLY is in effect for the input file, an output file may not be specified.

If the VIEWONLY control is specified, the input file may not be changed. The following function keys are not valid with VIEWONLY: <RUBOUT>, <DELCH>, <DELL>, <DELR>, <DELLI>.

The following commands are still displayed on the prompt line; however, they are not valid with VIEWONLY: (?)REPLACE, BLOCK Delete, GET, HEX Input, INSERT, MACRO Insert, MACRO Save, PARAGRAPH, QUIT Exit, QUIT Update, QUIT Write, and XCHANGE.

All other commands are legal. Note that you may save a VIEWONLY file or a portion of it using the BLOCK Put command.

While the VIEWONLY control is in effect, the message line displays the word View.

Error. "illegal command" is displayed if an illegal command is given.

4.2.3 RECOVER

Syntax

```
RECOVER | NORECOVER
```

Abbreviations

```
[NO]RC
```

Default

```
NORECOVER
```

Control Type

Processing mode for the main input file on invocation

Description

The RECOVER option can be used to help you reconstruct edited files if a fatal system error occurs during AEDIT-86 operation or if an unintentional termination of an AEDIT-86 session occurs using QUIT Abort.

If a crash occurs, reinvoke AEDIT-86 with the RECOVER control. The RECOVER control can be specified only for the main input file and only in the invocation line.

When RECOVER is specified, AEDIT-86 takes the entire memory contents as the input file. If the memory contains previously edited file(s), your file must be reconstructed. RECOVER is, however, only a means for the reconstruction—you must identify, gather, and connect the relevant text portions in memory.

RECOVER may be used only if the memory allocated to AEDIT-86 in the current activation is the same as that used in the previous activation. This implies that RECOVER is probably useless on virtual-memory-based systems or in a multitask environment.

The reconstruction process is difficult or impossible if the edited file is so large that it is spilled to extra memory or to temporary files. The memory content in such a case does not reflect the entire file contents.

If an input file is specified, the input file is not read when RECOVER is in effect, but it serves as an output file for the QUIT Update command.

4.2.4 MACRO

Syntax

MACRO [(*macro_file*)] | NOMACRO

Abbreviations

[NO]MR

Default

MACRO (AEDIT_*filename*.MAC)

Control Type

Execution mode

Description

The MACRO control allows you to specify a macro file for the current AEDIT-86 invocation. The NOMACRO option prevents AEDIT-86 from reading a macro file. Not specifying this control or just specifying MACRO is equivalent to the default. When the MACRO control is specified with a filename, the filename can have any extension. The subject of macro files is more thoroughly described in Part II.

4.2.5 MACROSIZE

Syntax

```
MACROSIZE ( macro_buffer_size )
```

Abbreviations

```
MS
```

Default

```
MACROSIZE ( 3072 )
```

Control Type

Execution mode

Description

This control is useful to allocate more macro buffer space if, for example, a huge batch operation is implemented using macros. Also, more macro space may be required if many or long macros are used. The subject of macros and macro files is more thoroughly described in Part II.

MACROSIZE allows you to specify the macro buffer size for the current AEDIT-86 invocation. *macro_buffer_size* is a decimal number specifying the number of bytes to be allocated. The minimum *macro_buffer_size* is 1024 (400H) bytes; the formal maximum allowed is 32767 (8000H) bytes. The default size is 3072 bytes. The maximum size actually allowed also depends on the amount of RAM available; therefore, it may be less than the formal maximum.

Buffer that is allocated for macros is not available for text; therefore, allocating a large macro buffer is not recommended.

4.2.6 BATCH

Syntax

```
BATCH | NOBATCH
```

Abbreviations

```
[NO]BA
```

Default

```
NOBATCH
```

Control Type

Execution mode

The BATCH control is used to activate AEDIT-86 in a noninteractive mode, usually from a command file. When BATCH is in effect, AEDIT-86 suppresses all output except the MESSAGE line.

Although AEDIT-86 may receive input from the console in BATCH mode (implying a semi-batch mode where input is from the keyboard), this is not recommended. For example, in this mode yes/no questions (e.g., "ok to replace?") are suppressed; however, AEDIT-86 still waits for the answers.

Activation with BATCH Control

If AEDIT-86 is invoked from a command file, all input is from the command file. The sequence of commands and characters should be exactly the same as if you were executing AEDIT-86 interactively. Input is echoed to the system console.

Example 1. If you want to change *dog* to *cat* throughout your file, you can create the following command file:

```
. . .
AEDIT EXAMPL.SRC BATCH
/Rdog<ESC>cat<ESC>QE
. . .
```

where

EXAMPL.SRC	is the input file.
/R	means replace all occurrences.
QE	is the QUIT Exit command.

< ESC > is inserted in the command file using the HEX Input command or using the < HEX > function key under INSERT.

Example 2. The text in example 1 was entered on one line because < cr > and < lf > are read as part of the command input sequence. < cr > is a legal command but < lf > is not and generates an error message.

To avoid this problem, an AEDIT-86 macro file may be created, e.g., EXAMPL.MAC. This macro file disables the character pair < cr > < lf > with the configuration command AFIG. The configuration command AFIG sets characters to be ignored (configuration commands are described in Chapter 9).In this case, you would set AFIG=0D0A.

In this example, the following command file may be used:

```
. . .
AEDIT EXAMPL.SRC BATCH
MGEXAMPL.MAC<ESC>
/Rdog<ESC>cat<ESC>
QE
. . .
```

where

EXAMPL.SRC	is the input file.
MG	is the MACRO Get command.
EXAMPL.MAC	is the macro file containing the configuration command AFIG=0D0A.
/R	means replace all occurrences.
QE	is the QUIT Exit command.

Example 3. If all operations are defined in a macro (creating macro files is described in Chapter 5), the command file requires only two AEDIT-86 commands. In this example, given a macro with the required operation sequence called PASS1 and a macro file called PASS1.MAC, the command file is as follows:

```
. . .
AEDIT EXAMPL.SRC BATCH
MGPASS1.MAC<ESC>EPASS1<ESC>
. . .
```

where

EXAMPLE.SRC	is the input file.
MG	is the MACRO Get command.
PASS1.MAC	is the macro file.
E	is the EXECUTE command.
PASS1	is the file containing the operation sequence.

4.3 Various BATCH Modes

The BATCH control can be either BATCH or NOBATCH. NOBATCH is the default. The activation source can be either the keyboard or a command file.

NOBATCH with input from the keyboard is the usual interactive mode. BATCH with input from a submit file is the usual batch operation. The remaining combinations, NOBATCH with input from a submit file and BATCH with input from the keyboard, are seldom used.

4.4 Banner

When AEDIT-86 is invoked, it reads and processes the macro file (if any), reads the input file (if any), clears the screen, and enters main command level. If an error is detected, an error message is displayed on the message line. Otherwise, the following banner is displayed:

```
---- system-id AEDIT Vx.y Copyright yyyy Intel Corp.
```

where:

----	is the busy/waiting indicator.
system-id	is supplied by the operating system.
x.y	is the AEDIT-86 version number.
yyyy	is the copyright year(s).

4.5 Work Files

The file :WORK: is used by AEDIT-86 as a predefined filename. AEDIT-86 assumes that any I/O operation to this file will be successful. It is your responsibility to make sure that :WORK: is available. System specific information on :WORK: is given in the appendixes.

PART II

Part II describes the more advanced uses of AEDIT-86. In this part, the macro commands and macro files, AEDIT-86 variables, the CALC command, advanced usage of AEDIT-86, and configuration commands are explained in detail.

Chapter 5, "Macro Commands," discusses the manipulation of macros and how to create and use them. Macros, sequences of commands that have been collected and given a name, are typically used for command sequences that are executed often.

Chapter 6, "AEDIT-86 Variables," describes the set of local and global variables that you can access.

Chapter 7, "CALC Command," describes the CALC command's capabilities. A CALC command expression can include logical operators, relational operators, shift/rotate operators, and arithmetic operators.

Chapter 8, "Advanced AEDIT-86 Usage," describes writing macros that incorporate the CALC command and AEDIT-86 variables.

Chapter 9, "Configuration Commands," gives the characteristics and configuration commands for the Series III and Series IV systems and the DEC VT100 family of terminals.

CHAPTER 5
MACRO COMMANDS

AEDIT-86 macros are sequences of AEDIT-86 commands (sequences of keystrokes) that have been collected and given a name.

Macros are typically used for frequently executed command sequences. Instead of entering a series of commands, you can call a previously created macro to execute a command sequence automatically. Macros speed your work and reduce the typing errors associated with long command sequences.

A set of macros may be grouped in a macro file. When you "get" such a macro file, all macros in that file are available for execution.

Macro files contain macro definitions and may contain configuration commands, SET commands, and comments.

AEDIT-86 enables you to create macros interactively using the MACRO Create command or directly using the MACRO Insert command. Typically, macros are defined interactively using MACRO Create, and corrected or changed using MACRO Insert.

Macros are stored in macro form. When you create a macro, letters and digits are entered as is. Function keys are stored in a special format: for example, the AEDIT-86 command < DELR > is stored in its macro form, in this case, \XA. (Control codes are listed later in this chapter.)

5.1 MACRO Command

The MACRO command is invoked by pressing M, which allows you to manipulate macros.

A Macro definition is a series of commands written in macro form. Macros can be defined in two ways: interactively, by using the MACRO Create command, or directly, by writing macros to your macro file using the MACRO Insert command, which inserts text in macro form automatically. To save interactively defined macros, you must write them to a separate macro file in macro form. The MACRO Get command is used to get a macro file, thus making the macros it contains available for execution.

5.1.1 Description

Press M; AEDIT-86 prompts—

```
Create  Get  Insert  List  Save
```

To specify a subcommand, press the initial letter of that subcommand.

Create

The Create subcommand "creates" a macro interactively by accumulating a sequence of keystrokes. The macro is executed and created concurrently.

Press C; AEDIT-86 prompts—

```
---- <HOME> to re-edit
Macro name:
```

The name of the last macro specified for MACRO Create, MACRO Save or EXECUTE (if any) is displayed to the right of the colon. Type in the macro name followed by <ESC> or <RETURN>.

A macro name can consists of either a single character or a character string of up to 60 characters. The macro name may contain any characters, e.g., +, 6, a.

After you type the macro name, the word "Macro" is displayed on the message line and remains there until the macro definition is complete. Then AEDIT-86 returns to main command level, and the entire set of AEDIT-86 commands is now available for MACRO Create.

All subsequent keystrokes are executed in the regular manner, but they are also "remembered" or "trapped" by the editor. These keystrokes constitute the macro definition, including special keys like <ESC> and <HEX>, which behave as usual for the current activation; however, they are also inserted into the macro definition. The macro is terminated by typing one of the following characters:

<CTRL-C>	Terminates MACRO mode without defining the macro; the macro is deleted.
M (main level only)	Successful termination of macro definition.

By defining a single-character macro, you are able to configure keys (see the EXECUTE command later in this chapter).

Following is an example that interactively creates the macro dot (.) that finds the next occurrence of the last target string:

```
M(ACRO) C(reate)
Macro name: .<ESC>
F<ESC>M
```

Following is an example that interactively creates a macro to configure <CTRL-L> to mean "jump to start of line."

```
M(ACRO) C(reate)
Macro name: <CTRL-L><ESC>
<RIGHT><LEFT><HOME>M
```

Error. "no more room for macros" is displayed on the message line, if macros exceed the amount of memory allocated to macros. In effect, the definition is terminated, and the current incomplete macro definition is deleted.

Get

The Get subcommand gets a macro file. This means that—

- The macro definitions in the file are available for execution.
- The configuration commands in the file are executed.
- The SET commands in the file are executed.
- The macro comments in the file are ignored.

Macro files are described later in this chapter.

The new macro definitions are added to the current set of available macros. If a macro in the new set has the same name as a macro already available, the new macro overrides the previous one. Configuration commands and SET commands are executed.

Press G; AEDIT-86 prompts—

```
---- <HOME> to re-edit
Macro file:
```

The name of the last macro file read (if any) is displayed to the right of the colon. Edit the macro filename, if required, then press ‹ RETURN ›.

You may insert an empty string as a filename by typing M(ACRO) G(et) ‹ RETURN › to instruct AEDIT-86 to "get" the present text buffer as a macro file.

When AEDIT-86 is invoked, the editor processes the initial macro file. The name of this file is taken from the macro control or, if this control is not specified, AEDIT-86 changes the invocation file extension to ‹ *root* ›.MAC and processes it automatically. This means that you do not have to get the default macro file to execute the macros in it. However, you must get any other macro file before its macros are available for execution.

Error. Any of the errors listed in Section B.4 may be issued during a MACRO Get. The error is displayed, the area causing the error is skipped, and processing continues.

Related Commands. The invocation controls MACRO, MACROSIZE

Insert

The Insert subcommand causes all subsequent input, including function keys (e.g., ‹ ESC › ‹ RUBOUT ›) to be inserted in the text in macro form and not executed. It is used to change and correct macro files. For example, if you press ‹ UP › in MACRO Insert, the character sequence \CU is inserted in the text. The macro definition may then be saved in a macro file. This command is terminated only by typing ‹ CTRL-C ›.

Press I; AEDIT-86 prompts—

```
Control C to stop
```

In MACRO Insert, all keys are entered as is (e.g., F(IND) is entered as F). Thus, keys such as ‹ HEX › or ‹ ESC › do not perform a function but are inserted as their macro codes. The following are exceptions:

- When ‹ RETURN › is typed, it is not converted to \NL because the carriage return is used to break macro definitions into more readable lines. Therefore, you must type \NL if ‹ RETURN › is required in the definition.
- If the backslash is not a lead-in character, it must be entered twice (\ \). However, the backslash is not doubled when it is typed, which enables you to type \MM to terminate the macro or \NL for ‹ RETURN ›.

Type ‹ CTRL-C › to terminate MACRO Insert mode.

Example. The following macro defines ‹ CTRL-L › to mean jump to start of line. (Remember that what you type does not execute but is inserted in macro form.)

EDIT *filename*
M(ACRO) I(nsert) \M<CTRL-L><ESC><RIGHT><LEFT><HOME> \MM (end macro) <CTRL-C>
Q(UIT) U(pdate) or Q(UIT) E(xit)

The following text is inserted into *filename*:

```
M\00C\BR\CR\CL\CH\MM
```

MM terminates the macro definition (see also Section 5.3, "Macro Modes").

List

The List subcommand displays on the message line the names of all currently available macros. If there are more macros available than can be listed on the message line, the message "hit space to continue" is displayed. Press the space bar to continue; any other character returns the editor to main command level.

Save

The Save subcommand translates an available macro to macro form and inserts the definition at the current position in the text. The macro may subsequently be modified or saved in a macro file. If you want to look at a macro definition, use MACRO Save to translate and display the macro, review it, and delete it (if desired).

Press S; AEDIT-86 prompts—

```
---- <HOME> to re-edit
Macro name:
```

The name of the last macro specified (for MACRO Create, MACRO Save or EXECUTE) is displayed to the right of the colon. Type a macro name followed by <ESC> or <RETURN>. If the macro exists, it is inserted in the text at the current cursor location in macro form.

You may use the following procedure to save a new, interactively created macro for future use:

1. Press O to enter the OTHER buffer.
2. Use the QUIT Init command to invoke your macro file.
3. Specify the macro filename when prompted.
4. Insert the macro in macro form using the MACRO Save command.
5. Update the modified macro file using the QUIT Update or QUIT Exit command.

5.2 Deleting Macros

To delete a macro from the set of available macros, use the MACRO Create command. Type the following:

```
M(ACRO) C(reate) macro_name<ESC><CTRL-C>
```

where

macro_name	is the name of the macro to be deleted.

This procedure does not delete a macro from a macro file. To delete from a macro file, you should edit the macro file like any other file, and use the delete commands.

5.3 Macro Modes

A macro may be either modeless (terminated with \MM after it is converted to macro form) or non-modeless (terminated with \EM after it is converted to macro form). All macros created with MACRO Create are modeless. A non-modeless macro may be created using MACRO Insert or by editing a saved modeless macro.

Any modeless or non-modeless macro may be used at main command level or in either INSERT or XCHANGE modes.

A modeless macro is independent of whether it is called from main command level, INSERT or XCHANGE mode. This allows you to use the same macro at the main command level and in INSERT or XCHANGE mode. When you execute a modeless macro, it executes as if it is at main command level. When it finishes execution, it restores the mode (e.g., XCHANGE) that was in effect when it was activated.

A non-modeless macro is executed at the AEDIT-86 prompt level that was in effect when the macro was activated. When the macro finishes execution, it does not restore the mode that was in effect when it was activated. Instead, AEDIT-86 remains in the mode determined by the macro. Non-modeless macros provide compatibility and upgrading with respect to AEDIT V1.0.

Modeless macro execution always give the same results regardless of the mode from which it was executed. Non-modeless macro execution results usually depend on the context (mode) from which they were called. It is recommended that you use modeless macros whenever possible.

For example, compare the following macros:

```
MA\BRi*\BR\MM
MB\BRi*\BR\EM
```

The first macro is modeless. When it is executed, the character * is inserted whether or not it was called from main command level, INSERT, or XCHANGE mode; when the macro finishes, the initial mode is retained.

The second macro is non-modeless. If it is called from main command level, the results are the same as for the first; i.e., the character * is inserted and the editor remains at main command level. But, if the second macro is called from INSERT mode, for example, the characters i* are inserted into the text, and the ‹ESC› command (\ BR) causes the editor to leave INSERT mode and return to main command level.

5.4 EXECUTE Command

The EXECUTE command requests a macro name and executes the specified macro.

In macro execution all input is taken from the macro except for answers to the following questions/requests:

- ?Replace: "ok to replace?"
- QUIT Init or QUIT Abort: "all changes lost?"
- BLOCK Delete: "cannot save in memory, save anyway?"
- "hit space to continue"
- QUIT Write or BLOCK Put: "overwrite existing file?"

In the prompts listed above, the response to the prompt is taken from the console.

5.4.1 Description

Press [*count*] E; AEDIT-86 prompts—

```
---- <HOME> to re-edit
Macro name:
```

The name of the last macro specified (for MACRO Create, MACRO Save, or EXECUTE) is displayed to the right of the colon. Type a macro name followed by < ESC > or < RETURN >. If the macro exists, it is executed.

If the busy/waiting indicator is active, the prompt line displays -!!- when a macro is being executed. This is important in single-character macro execution where there may not be any other indication that a macro is still executing.

The macro terminates when it has been executed the specified number of times or failed. Section 5.8 describes macro execution termination in more detail.

Macro activation may be nested up to eight levels.

Type a < CTRL-C > to force termination of macro execution.

Errors

- "no such macro" is displayed if the macro specified does not exist.
- "macro nesting too deep" is displayed if you attempt to nest macros to more than eight levels.

Count. The EXECUTE command accepts any count.

Related Commands. SET Display, SET Go

5.5 Macros and AEDIT-86 Variables

When AEDIT-86 variables are referenced during MACRO Create, the actual variable value is fetched for the current activation of the macro, and a reference to this variable is inserted into the macro definition. The use of AEDIT-86 variables provides a primitive way to simulate passing a parameter to a macro. See Section 6.3 for more details.

5.6 Single-Character Macros

You can write macros with single-character names. You can activate a single-character macro by simply typing its name if this character has no other function in the context from which it is being activated. Single-characters that can be used as macro names are referred to as "free" characters, e.g., L, U, Z, +, !.

Single-character macros can be activated in the following ways:

- Using the EXECUTE command at main command level (as in the case for all macros).
- By pressing the macro character preceded by the < MEXEC > key.
- By pressing the key itself, if the key is "free," that is—

  In INSERT and XCHANGE modes: all nonconfigured control characters. The following control characters also cannot be used as macros in this way: < CTRL-M > (< RETURN >), < CTRL-I > (< TAB >), and < CTRL-J > (line feed).

  In main command levels: the same as for INSERT and XCHANGE modes with the addition of all printable characters that are not used as AEDIT-86 commands (e.g., +, $, U, etc.).

<MEXEC> is a configurable key (usually <CTRL-E>). To activate the single-character macro, press <MEXEC> followed by the macro name. For example, if you are using the macros from USEFUL.MAC in INSERT mode, typing <MEXEC> ↑ converts the word that starts "under" the cursor to uppercase letters.

A digit may not be used as a single-character macro at main command level because it is always interpreted as a count. A function key may not be used as a macro name because the key's function overrides the macro definition.

The following is an example to demonstrate using a single-character macro in INSERT or XCHANGE mode. You can define a macro called <CTRL-P> as the word PROCEDURE and save this macro in a file called PLM.MAC. If you are editing a PL/M source file, you can call PLM.MAC with MACRO Get; then, each time you type <CTRL-P> the word PROCEDURE will be inserted into the file. This saves having to type out the word each time you want to insert it.

Errors

- "illegal command" is displayed when you type a character that is not a command abbreviation, a decimal digit, or a macro name, at main command level.
- "no such macro" is displayed if the character following the EXECUTE command or the <MEXEC> key is not a macro name; i.e., no macro with that name exists.

5.7 Macro Files

A macro file may consist of the following:

- Configuration commands
- SET commands
- Macro definitions
- Macro comments

Configuration commands specify the host/terminal characteristics (described in Chapter 9).

SET commands are the only commands, with the exception of configuration commands, that can be specified in a macro file. Including SET commands allows you to specify the mode of operation. For example, if you use a macro called AEDIT.MAC to "set" your environment, you may include the command SET K_token Yes if you want the environment initialized to search for tokens only. In the macro file, this command appears as SKY.

A macro definition is a series of commands written in macro form. It has the following format:

```
M macro_name \BR characters_in_macro \MM
```

where

M	declares that a macro definition follows.
macro_name	is any name given to the macro being defined.
\BR	stands for <ESC>.
characters_in_macro	is the macro contents.
\MM	signals the end of a modeless macro.
\EM	signals the end of a non-modeless macro.

The following representations of control characters and control codes are used in the macro definitions.

Name	Represents
\BR	< ESC >
\CL	< LEFT >
\CR	< RIGHT >
\CU	< UP >
\CD	< DOWN >
\CH	< HOME >
\NL	< RETURN >
\RB	< RUBOUT >
\TB	< TAB >
\XF	< DELCH > Delete Character
\XX	< DELL > Delete Left
\XA	< DELR > Delete Right
\XZ	< DELLI > Delete Line
\XU	< UNDO >
\XH	< HEX > Hex prefix character
\XE	< MEXEC > Macro Execute
\XN	< FETN > Fetch Numeric
\XS	< FETS > Fetch String
\0*h*	Hexadecimal value of a character
\MM	End of modeless macro definition
\EM	End of non-modeless macro definition

The blackslash (\ ) must appear twice if it is not used as a code lead.

A macro definition and configuration commands should be ended with either a semicolon (;) or < cr >. < cr > and < lf > (as opposed to \NL and \0A) are ignored in a macro file, even within macro definitions. This allows you to split the macro definition into lines so that it is easier to read.

Comments may be included in a macro file. A comment starts with a backslash/asterisk character pair (\ *) and ends with an asterisk/backslash pair (* \ ). All characters, including carriage return and line feed, may appear in the comment. The following is an example of a comment:

```
\* this is a comment *\
```

5.8 Macro Execution after a Failure

A command execution is marked as failed if—

- An attempt is made to move forward (< RIGHT >, < DOWN >, < HOME >, or < RETURN >) at the end of the file.
- An attempt is made to move backward (< LEFT >, < UP >, or < HOME >) at the start of the file.
- A (-)FIND or (?)REPLACE command fails to find the target string.

A command prefixed by a finite count is marked as failed if any of its execution is thus marked.

A (-)FIND or (?)REPLACE command prefixed by / is marked as failed only if it fails on its first execution.

A cursor movement command prefixed by / is never marked as failed.

If the SET Go option is on, the (-)FIND and (?)REPLACE commands are never marked failed.

When a command in a macro is marked as failed, the following occurs: Macro execution is terminated, and control is returned to the caller. If the caller is main command level, AEDIT-86 simply waits for the next command. If the caller itself is a macro, execution continues with the caller's next command.

Example

```
Macro A:    /E(XECUTE) B
            E(XECUTE) C
            E(XECUTE) D

Macro B:    J(UMP) S(tart)
            /R(EPLACE) "<nl> <nl>" with "<nl>"

Macro C:    S(ET) G(o) Y(es)
            J(UMP) S(tart)
            /R(EPLACE) "DCL" with "DECLARE"
            J(UMP) S(tart)
            /R(EPLACE) "IS" with "LITERALLY"
```

- A value for SET Go in macro A is meaningless, because it does not contain a (-)FIND or (?)REPLACE command. When /EXECUTE B is terminated either normally or because macro B failed, macro C is executed. Likewise, when macro C is terminated, macro D is executed.
- SET Go must be set to No (the default) for macro B. The REPLACE command is successful as long as the file contains at least one <nl> <nl> sequence. When the file contains no <nl> <nl> sequences, macro B fails, execution of macro B is terminated, and macro C is executed. If SET Go is Yes for macro B, it will never fail, and execution of macro B would enter an infinite loop.
- SET Go must be set to Yes for macro C. This option ensures that the second REPLACE command is performed regardless of the results of the first REPLACE command. S(ET) G(o) Y(es) could be placed after the JUMP command, and the effect would be the same.

If you are not careful in coding your macro, it might enter an infinite loop when it executes. To exit from such a macro or to terminate any macro, press <CTRL-C>.

5.9 Screen Display during Macro Execution

To speed up macro execution, the amount of data written to the screen while a macro is executing is reduced to a minimum. Only selected text or information is sent to the screen. The information in this section is given for reference.

5.9.1 Text

If SET Display No (the default) is in effect, changes in the text and cursor movements are displayed on the screen as long as the cursor does not leave the current display screen, or until a VIEW command (either explicit or implicit) is issued. When the cursor leaves the screen or a VIEW command is issued, screen display is frozen until the macro execution terminates. Then, an implicit VIEW command is performed using the current logical cursor location.

If SET Display Yes is in effect, all changes and cursor movements are displayed on the screen, even if the cursor leaves the screen or a VIEW command is given.

Regardless of the SET Display value, the text is updated if you give the ?REPLACE command or the FIND/REPLACE command and SET Showfind Yes is in effect.

5.9.2 Message

The message line is updated only in the following cases:

- Error message display. The error message display lasts for about a second. The FIND/REPLACE command message "not found" is not considered an error.
- CALC command messages. For an argument that is an expression rather than an assignment statement (e.g., N3+1 versus N2=N3+1 or S9 versus S9="abc").
- HEX Output and MACRO List command messages.
- QUIT or OTHER command filename messages.

The above set is carefully chosen to contain the cases where either the message is important or the command affects the message line only.

When macro execution is terminated, the last message remains on the screen. However, the status (e.g., Other, Viewonly) is updated if needed.

The "found: *count*" message is a special case; it is displayed only if it is the last message of the macro execution at main command level.

5.9.3 Prompt

The prompt line is changed only when a macro requests an answer to one of the questions listed in Section 5.4.

After macro execution terminates, the prompt line reflects the current mode of the editor. This mode is either main command level, or INSERT or XCHANGE mode.

5.9.4 Window

When a new window is constructed, the text is updated immediately, which is an exception to the previous description. The reason for this exception is to create a place for a future message that may refer to the upper window. The KILL_WND command operates in the usual manner. That is, when the cursor leaves the screen, the screen will be updated only when macro execution terminates.

5.10 Macro Examples

5.10.1 Example 1

It is often necessary to reset the left column. The single-character macro right square bracket (]) sets the left column one position to the right each time it is executed, and the single-character macro left squarebracket ([) sets the left column one position to the left each time it is executed:

```
M ] \BRSL+1 \NL \MM
M [ \BRSL-1 \NL \MM
```

These macros can be defined interactively using the MACRO Create command. For example, the first macro can be created by entering the following commands:

```
M(ACRO) C(reate)
Macro name: ] <ESC>
S(ET) L(eftcol) +1 <RETURN>
M (to terminate macro definition)
```

After you define these macros, typing a right square bracket (]) at main command level or ‹MEXEC›] in INSERT or XCHANGE mode sets Leftcol one position to the right of its current setting; typing a left square bracket ([) at main command level or ‹MEXEC› [in INSERT or XCHANGE mode sets Leftcol one position to the left of its current setting.

5.10.2 Example 2

The following single-character macros, named dot (.) and comma (,) find the next occurrence of the target string, or find the previous occurrence of the target string, respectively.

```
M . \BRf \BR \MM
M , \BR- \BR \MM
```

These macros can be defined interactively using the MACRO Create command. For example, the first macro could be created as follows:

```
M(ACRO) C(reate)
Macro name: . <ESC>
F(IND)<ESC>
M (to terminate the macro definition)
```

After you define these macros, typing a dot (.) at main command level or ‹MEXEC› followed by a dot in INSERT or XCHANGE modes finds the next occurrence of the target string; typing a comma (,) at main command or ‹MEXEC› followed by a comma in INSERT or XCHANGE modes, finds the previous occurrence of the target string.

5.10.3 Example 3

It is often desirable to use visual breakpoints in programs. For example, you can use comment lines filled with hyphens to separate procedures in a language. The following macro creates a break line:

```
M@ \ BRI(* - - - - - - - - - - - - - - - - - - - *) \ NL \ BR \ MM
```

You can create this macro interactively with the following commands:

```
M(ACRO) C(reate)
Macro name: @ <RETURN>
I (* - - - - - - - - - - - - - - - - - - - - - - *)<RETURN><ESC>
M
```

After you define this macro, typing @ at main command level, or ‹MEXEC›@ in INSERT or XCHANGE mode inserts the break line at the current cursor position in text.

5.10.4 Example 4

The following sequence of commands saves, in a new file named EXMPL.MAC, an interactively defined macro named asterisk (*) that allows you to scroll backwards ten lines.

M(ACRO) C(reate)	Create macro * interactively
`Macro name: * <RETURN>`	
`10<UP>M`	Command sequence
O(THER)	
Q(UIT) I(nit) `EXMPL.MAC<RETURN>`	
M(ACRO) S(ave)	Invoke the MACRO Save command, which prompts for the macro name
`Macro name: * <RETURN>`	Execute the MACRO Save command, translate the macro * to macro form.
Q(UIT) U(pdate)	Update the macro file (EXMPL.MAC), which now includes the macro *.

CHAPTER 6
AEDIT-86 VARIABLES

AEDIT-86 has a set of variables that can be accessed by the user. This set has the following characteristics: string variables versus numeric variables, read-only variables versus read-write variables, local variables versus global variables.

- Read-only variables reflect internal AEDIT-86 values that you can retrieve but not modify. Read-write variables can be modified freely. Read-write variable assignment can be done only in the CALC command.
- Local variables can be accessed only in the CALC command. Global variables can be accessed in other contexts as well.

6.1 Global Variables

The two types of global variables are numeric and string. The global numeric variables are all read-write. The global string variables can be read-only or read-write.

6.1.1 Global Numeric Variables

The ten global numeric read-write variables (N-variables), are N0–N9, which are 32-bit numbers. Values are assigned to the N-variables only in the CALC command. To fetch an N-variable, type < FETN > *i*, where < FETN > is the "fetch numeric" key (usually configured to < CTRL-N >), and *i* is any digit from 0–9. When AEDIT-86 is invoked, the N-variables are initialized to zero.

The N-variables may be used in the following ways:

- In any line-edit prompt, e.g., *target-string*, *replacement-string*, or GET *filename*. The contents are inserted and displayed as signed decimal numbers; leading zeros are suppressed. The conversion is carried out by the line-editing mechanism, regardless of the command currently being executed.

  Note that a confusing case occurs when the hexadecimal value of a variable is interpreted as a decimal value. For example, if N1 contains the hexadecimal value 2DH, which is equivalent to 45 decimal and you type < FETN > 1 under HEX Input, the character E (45H ASCII) is inserted into the text.
- As a count (or part of a count) for a command. Count cannot be negative; therefore, the absolute value of the N-variable is used. In this case, the value of the N-variable should be in the legal range, 0–65535. The contents are displayed as an unsigned decimal number; leading zeros are suppressed.
- In INSERT and XCHANGE modes. For example, if < FETN > 1 is typed, the contents of N1 are inserted in the text in a form that depends on the value of SET Radix. For example, if variable N1 contains the hexadecimal value 45H, then—

  IF SET Radix is alpha: E
  IF SET Radix is binary: 1000101
  IF SET Radix is decimal: 69
  IF SET Radix is hex: 45
  IF SET Radix is octal: 105

  The value is inserted without a suffix and leading zeros are suppressed.

- Under the CALC command. The variable may be retrieved as in any line-edit prompt. Also, the variable name may be used as is, e.g., N1 instead of ‹FETN› 1. In this case the name, rather than the value, is displayed, e.g., N1 instead of 45. The entire processing is done by the CALC command, not by the line-editing mechanism. A variable may be modified only if it appears in CALC with its name on the left-hand side of an assignment statement.

  When the ‹FETN› key is pressed, the message ‹FETN› appears on the message line. This message disappears when the next key is pressed.

6.1.2 Global String Variables

The two groups of global string-variables (S-variables) are—

- Read-write string variables
- Read-only string variables

Read-Write String Variables

The ten read-write variables are S0–S9. Value assignment is done only in the CALC command. When AEDIT-86 is invoked, these variables are initialized to a null string.

Read-Only String Variables

The following is an alphabetic list of the read-only variables. No assignment of values is allowed.

SB	Up to 60 characters of the Block buffer. By using SB, a portion of the text file may be used later as, for example, an argument in a FIND command.
SE	The name of the current edited file (as opposed to the secondary file).
SG	The name of the last file specified in the GET command.
SI	The name of the main input file.
SM	The name of the last file specified for the MACRO Get command.
SO	The name of the OTHER input file.
SP	The name of the last file specified for the BLOCK Put command.
SR	The replacement string of (?)REPLACE.
ST	The target string of (-)FIND and (?)REPLACE.
SW	The name of the last file specified in the QUIT Write command.

Using String Variables

To fetch the value of a string variable, type ‹FETS› *x*, ‹FETS› is the "fetch string" key (usually configured to ‹CTRL-S›), and where *x* is a digit (0–9) or a letter appearing as the second letter in a name in the above list (e.g., B, E, G). For example, ‹FETS› 7 fetches the value of S7, and ‹FETS› G fetches the value of SG. An S-variable contains a string of 0–60 characters.

The S-variables may be used in the following circumstances:

- In any line-edit prompt, e.g., *target-string*, *replacement-string*, GET *filename*. The contents are inserted and displayed as an ASCII string. The conversion from ‹ FETS › *x* to the ASCII string is done by the line-editing mechanism, regardless of which command is currently being executed.
- In INSERT and XCHANGE modes. For example, if ‹ FETS › 1 is typed, the contents of S1 are inserted in the text.

  Note that a character that is inserted in this way loses any special meaning it may otherwise have. For example, 0DH is inserted as is and not as a carriage return, or 01BH (‹ ESC ›) does not cause mode termination.
- Under the CALC command. The S-variable may be retrieved as with any line-edit prompt. Also, the S-variable name may be used as is, e.g., SB instead of ‹ FETS › B. In this case, the name rather than the value is displayed, e.g., SM rather than AEDIT.MAC. All processing is done by the CALC command and not by the line-editing mechanism. An S-variable may be modified only if it appears in CALC with its name on the left side of an assignment statement.

An S-variable is always considered as if all its characters are literalized; i.e., they are interpreted as regular characters even if in other cases they may have a special meaning, e.g., ‹ HEX ›. Thus an S variable is never searched to determine if it fetches another S variable. However, in any other legal context, input is expanded if it includes a fetch operation of the form ‹ FETS › *x*.

When the ‹ FETS › key is pressed, the message ‹ FETS › appears on the message line. This message disappears when the next key is pressed.

6.2 Local Variables

All local variables are read-only numeric variables. Therefore, they cannot be assigned, and they can be used only in the CALC command.

The following is an alphabetic list of positional values:

BOF	Logical value—true if the cursor is at the beginning of file.
COL	The current logical cursor position in the line. (This value is not affected by the setting of Leftcol.)
CURCH	ASCII value of the current character.
CURWD	ASCII value of the two bytes at the current cursor location.
EOF	Logical value—true if the cursor is at the end of file.
INOTHR	Logical value—true, if you are in the OTHER buffer.
ISDEL	Logical value—true if the character at the current position is in the user defined delimiter set.
ISWHTE	Logical value—true if the character at the current position is a white space (space, tab, LF or CR).
LOWCH	If the current character is an uppercase character (41H to 5AH), LOWCH is the ASCII value of the lowercase character. Otherwise, LOWCH is the same as CURCH.

NXTCH	ASCII value of the next character.
NXTTAB	The column number of the next (i.e., to the right of the cursor) tab position as defined by SET Tab. If there are no tabs to the right of the cursor in the current line, the value of NXTTAB is zero.
NXTWD	ASCII value of the second and third bytes following the current cursor location.
ROW	Current cursor row (actual row, not the logical line in the text).
UPCH	If the current character is a lowercase character (61H to 7AH), UPCH is the ASCII value of the uppercase character. Otherwise, UPCH is the same as CURCH.

The following values are offset from the beginning of the currently processed input file. Note that these variables count the carriage return/line feed pair as two characters, whereas AEDIT-86 usually counts the carriage return/line feed pair as one character. If the file is edited using the FORWARDONLY control, the offset is from the current beginning of text. This position may vary as the cursor is moved forward.

CURPOS	Offset of current location in file. CURPOS is zero for the first character of the file.
TAGA	Offset of tag A.
TAGB	Offset of tag B.
TAGC	Offset of tag C.
TAGD	Offset of tag D.

The following values relate to the S-variables:

SL0–SL9	Relates to the S-variables S0–S9. Each variable contains the length of the corresponding global read-write S-variable.
SL*x*	Relates to the S-variables, S*x*, where *x* is the second letter in one of the global read-only variable names (e.g., B, E, G). Each variable contains the length of the corresponding global read-only S-variable.

The following is an alphabetic list of counters that contain the actual number of command repetitions from the last time the command was specified with count greater than one. The CNT prefix signifies COUNT.

CNTEXE	The number of times the macro that is currently executing has executed in the current activation. The first execution is number one. If none, the value is zero.
CNTFND	Relates to (-)FIND.
CNTMAC	The number of times that the last macro (which has finished executing) was executed.
CNTREP	Relates to (?)REPLACE.

The following values relate to the margin and indentation settings used by PARAGRAPH and SET Autonl commands:

IMARGN	The value of current indent margin setting.
LMARGN	The value of current left margin setting.
RMARGN	The value of current right margin setting.

The following values are returned by the UDI-call DQ$GET$TIME:

DATE	Date returned by DQ$GET$TIME in decimal format *mmddyy*.
TIME	Time returned by DQ$GET$TIME in decimal format *hhmmss*; note that some systems always return 0 for the time function.

The following are other values:

LSTFND	Logical value—true if the last find or replace string was found. (Note that an infinite FIND (/F) sets this variable to TRUE if at least one FIND succeeded.)
NSTLVL	Nesting level of the currently executing macro, console input is level 0.

6.3 Usage and Interpretation of Global Variables in Macros

When the value of any AEDIT-86 variable is fetched during MACRO Create, the actual value is used for the current activation. A reference to this variable is inserted into the macro definition. Therefore, in a future activation of the macro, the value of the variable at the time of macro activation will be used, not the value of the variable at the time of macro definition.

For example, if you have the following macro:

I(NSERT) < FETS > 7 < ESC >

and S7="*abc*" at the time the macro was defined, *abc* is used for the current activation. If the definition of S7 is changed to *xyz* and the above macro is again activated, *xyz* is inserted.

6.4 Summary of AEDIT-86 Variables

6.4.1 Read-Only Variables

The following is an alphabetic list of the read-only string variables. No assignment of values is allowed.

SB	Up to 60 characters of the Block buffer.
SE	The name of the current edited file.
SG	The name of the last file specified in the GET command.
SI	The name of the main input file.

SM	The name of the last file specified for the MACRO Get command.
SO	The name of the OTHER input file.
SP	The name of the last file specified for the BLOCK Put command.
SR	The replacement string of (?)REPLACE.
ST	The target string of (-)FIND and (?)REPLACE.
SW	The name of the last file specified for the QUIT Write command.

6.4.2 Local Variables

All local variables are read-only numeric variables. They cannot be assigned, and they can be used only in the CALC command.

The following is an alphabetical list of positional values:

BOF	Logical value—true if the cursor is at the beginning of file.
COL	The current logical cursor position in the line. (This value is not affected by the setting of Leftcol.)
CURCH	ASCII value of the current character.
CURWD	ASCII value of the two bytes at the current cursor location.
EOF	Logical value—true if the cursor is at the end of file.
INOTHR	Logical value—true if you are in the OTHER buffer.
ISDEL	Logical value—true if the character at the current position is in the user defined delimiter set.
ISWHTE	Logical value—true if the character at the current position is a white space (space, tab, LF or CR).
LOWCH	If the current character is an uppercase character (41H to 5AH), LOWCH is the ASCII value of the lowercase character. Otherwise, LOWCH is the same as CURCH.
NXTCH	ASCII value of the next character.
NXTTAB	The column number of the next (i.e., to the right of the cursor) tab position as defined by SET Tab.
NXTWD	ASCII value of the second and third bytes following the current cursor location.
ROW	The current cursor row (actual row, not the logical line in the text).
UPCH	If the current character is a lowercase character (61H to 7AH), UPCH is the ASCII value of the uppercase character. Otherwise, UPCH is the same as CURCH.

The following values are offset from the beginning of the currently processed input file:

CURPOS	The offset of current location in file.
TAGA	The offset of tag A.
TAGB	The offset of tag B.
TAGC	The offset of tag C.
TAGD	The offset of tag D.

The following values relate to the S-variables:

SL0–SL9	Relates to the S-variables S0–S9.
SL*x*	Relates to the S-variables, S*x*, where *x* is a second letter of one of the global read-only variable names (e.g., B, E, G).

The following is an alphabetic list of counters that contain the actual number of command repetitions from the last time the command was specified with count greater than one:

CNTEXE	The number of times the macro that is currently executing has executed in the current activation. The first execution is number one. If none, the value is zero.
CNTFND	Relates to (-)FIND.
CNTMAC	The number of times that the last macro (which has finished executing) was executed.
CNTREP	Relates to (?)REPLACE.

The following values relate to the margin and indentation setting used by PARAGRAPH and SET Autonl commands:

IMARGN	The value of current indent margin setting.
LMARGN	The value of current left margin setting.
RMARGN	The value of current right margin setting.

The following values are returned by the UDI-call DQ$GET$TIME:

DATE	Date in decimal format *mmddyy*.
TIME	Time in decimal format *hhmmss*; note that some systems always return 0 for the time function.

The following are other values:

LSTFND	Logical value—true if the last find or replace string was found.
NSTLVL	The nesting level of the currently executing macro; console input is level zero.

CHAPTER 7 CALC COMMAND

The CALC command provides you with computation capabilities. These capabilities coupled with AEDIT-86 variables enable the following, for example: centering a phrase on a line, finding the size of an input file, or changing a letter from lowercase to uppercase or vice versa.

To execute the CALC command, press C; AEDIT-86 prompts—

```
---- <HOME> to re-edit
Calc:
```

The last statement entered under CALC is displayed to the right of the colon.

7.1 Introduction

Either a numeric or a string statement is legal input to a CALC command. In general, the statement is either an assignment statement or merely an expression. A CALC expression is comparable to expressions in many other programming languages. It includes logical operators, relational operators, shift/rotate operators, and arithmetic operators. As usual, an expression may include parentheses. These, plus the precedence and associativity rules of the operators, completely define the evaluation order.

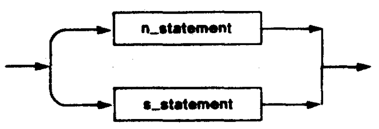
The operators act on the following elements: numeric constant (integers only), string constants, AEDIT-86 numeric local variables, N-variables, and S-variables.

Arithmetic using the CALC command is 32-bit signed arithmetic. No overflow detection is performed.

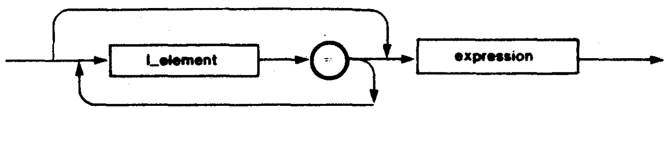
7.2 Syntax Diagrams for CALC Statement

This section presents the CALC input syntax in syntax diagram form. In these diagrams, every path you can follow in the direction of the arrow represents a syntactically correct construct of CALC input. Terminal symbols of the language, keywords, letter symbols, and punctuation symbols that are typed verbatim as CALC input are enclosed in circles or ovals. Nonterminal symbols (terms standing for language constructs that are defined in their own syntax diagrams) are enclosed in rectangles.

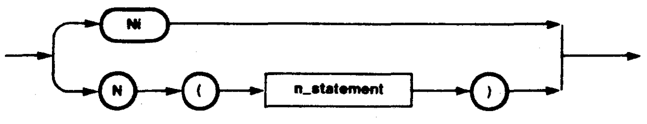
statement:



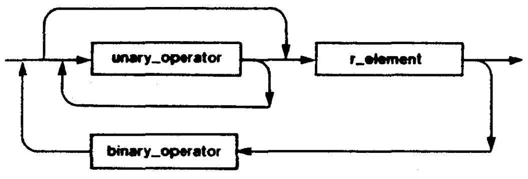
n_statement:



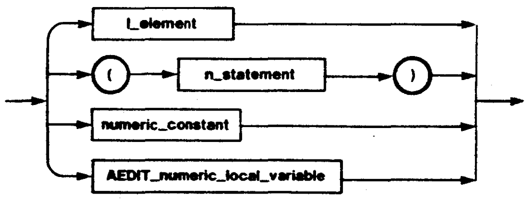
l_element:



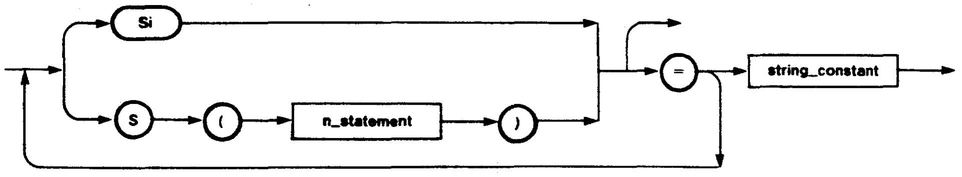
expression:



r_element:



s_statement:



Notes:

- n_statement, s_statement : *n* stands for numeric, *s* stands for string.
- l_element, r_element : *l* stands for left, *r* stands for right.
- Uppercase and lowercase letters are not distinguished from each other except in string constants. For example, curpos and CURPOS are interchangeable.
- One or more blanks may be inserted between tokens. However, when no ambiguity is possible, blanks are not required. For example, 1+2 is a legal expression.
- N*i*, S*i*, N(...), S(...) : the value of *i* and the value of the expression within the parentheses should be in the range 0–9.
- The value true is represented by −1; i.e., 0FFFFFFFFH, and the value false is represented by 0. Whenever an operator produces a logical (Boolean) value, that value is either true or false.
- unary_operator : see Table 7-1.
- binary_operator : see Table 7-1.
- AEDIT_numeric_local_variable : described in Chapter 6.
- numeric_constant : see Section 7.3.
- string_constant : see Section 7.4.

Table 7-1. Operators' Precedence and Associativity

Operator Class	Operator	Interpretation	Associativity
Parentheses	( )	Controls evaluation order: expressions in parentheses are evaluated before the action of any outside operator on the items in parentheses	From inside to outside
Unary	+ − - ` ! #	Single positive operator, Single negative operator, 1's complement (- or `) POS operator (!) NEG operator (#)	From right to left, e.g., !#3 is !(#3)
Power	**	Raising to the power of	From right to left, e.g., 3 ** 4 ** 5 is 3 ** (4 **5)
Arithmetic (mul/div)	* / \	Multiplication, division, mod (remainder) div	From left to right, e.g., 11 * 12 * 13 is (11 * 12) * 13
Arithmetic (add/sub)	+ −	Addition, subtraction	From left to right, e.g., 2−3+4 is (2−3)+4
Shift/rotate	SHL,SHR,SAL,SAR,ROL,ROR	Shift left, shift right, shift algebraic left, shift algebraic right, rotate left, rotate right.	From left to right, e.g., 5 SAR 1 SAR 1 is (5 SAR 1) SAR 1
Relational	< <= == <> => >	Less than, less than or equal to, equal to, not equal to, greater than or equal to, greater than	From left to right, e.g., 3 < 4 < 2 is (3 < 4) < 2
Logical	& | ^	AND, OR, XOR	From left to right, e.g., 5 & 3 ^ 7 is (5 & 3) ^ 7

7.3 Numeric Constants

The CALC command numeric constants are integer numbers. They can be binary, octal, decimal, or hexadecimal. The CALC command recognizes these constants by the suffix B, O (or Q), D, or H, respectively. Numbers without a suffix are considered decimal. A numeric constant may be in the range −(2**31) to +(2**31)−1.

7.4 String Constants

A string constant may be 0–60 characters long with same delimiter at both ends. The same delimiter means that there is no predefined string delimiter; rather, the character immediately to the left of the string constant is identified as a delimiter. Then the second occurrence of that character signifies the right end of the string. To prevent ambiguity, the following characters may not serve as string delimiters: letters, digits, blank, and tab.

A natural delimiter choice is a quotation mark. However, if the string constant includes a quotation mark, a different character, one that does not appear in the string, should be used as the delimiter.

The case of the letters within the string is preserved.

7.5 Operators

Syntactically, operators are divided into two groups: unary operators and binary operators. From the functional point of view, the operators are divided into the following groups: logical operators, relational operators, shift/rotate operators, and arithmetic operators. Table 7-1 partially reflects this partition.

Table 7-1 lists all the CALC command operators with a brief description of the semantics of each operator. Following are more detailed descriptions of the nontrivial operators:

1. ~ and ` (1's complement) have the same meaning. The duplicate notation prevents possible difficulties on terminals where one of these characters may have a special meaning. 1's complement means for every 0 bit, a 1 bit is returned and vice versa.
2. ! and # :

   The POS operator (!) is defined as follows:
   If number $\geq$ 0 then return true (-1)
   else return false (0).

   The NEG operator (#) is defined as follows:
   If number $\geq$ 0 then return false (0)
   else return true (-1).
3. \ (modulo division) returns the remainder of an integer division, for example, 7 \ 4 = 3; 16 \ 4 = 0.
4. $\wedge$ (XOR) returns true only if one operand is true and the other is false; otherwise it returns false. This is done for each bit in the argument, for example, $5\wedge1=4$.

7.5.1 Shift/Rotate Operators

In the shift/rotate operations, the left operand is handled as a pattern of 32 bits. It is moved to the right or to the left by the number of bits specified by the right operand.

In a shift, bits moved off one end of the pattern are lost, and 0 bits or 1 bits are moved into the pattern from the other end. In a rotate, bits moved off of one end move onto the other end.

SAL and SAR are algebraic shift operators. This means that the high order bit is the sign bit, and there is no shift of bits between the sign and the rest of the number. In a left shift (SAL), 0 bits move into the pattern from the right. In a right shift (SAR), either 0 bits (if pattern is positive) or 1 bits (if pattern is negative) move into the pattern from the left.

In every shift/rotate operation, the right operand (count) is always taken as modulo 256.

7.6 Expression Evaluation

Operators in the CALC command have an implied order that determines how operands and operators are grouped and evaluated.

Table 7-1 lists the CALC command operators from highest to lowest precedence (i.e., those that take effect first are listed first). Operators in the same line are of equal precedence.

The evaluation order is the same as that used in most programming languages. It is controlled first by parentheses, then by operator precedence, and finally by operator associativity. The CALC command first evaluates operands and operators enclosed in paired parentheses as subexpressions, working from innermost to outermost parentheses pairs. The value of the subexpression is then used as an operand in the remainder of the expression.

The precedence and associativity rules are demonstrates by the following examples:

```
n1 + n2 * n3 ** - n4                          is equivalent to
n1 + (n2 * (n3 ** (-n4)))

n1 SHR 3 ≥ n2 SHL n7 + 4 * #2                 is equivalent to
(n1 SHR 3) ≥ (n2 SHL (n7 + (4 * (#2))))

n2 + 3 < > n4 - 5 & n4 - 2 == n5              is equivalent to
((n2 + 3) < > (n4 - 5)) & ((n4 - 2) == n5)
```

7.7 Examples

See also Chapter 8 on Advanced AEDIT-86 Usage for examples using the CALC command.

7.7.1 Example 1

```
S3
```

The value of the S-variable S3 is displayed at the message line; no assignment.

7.7.2 Example 2

```
n1 = n ( n ( n2 = 2 * n3) )
```

Assuming that N3 contains 3 and N6 contains 8, and all other N-variables are zero. Then 2 * N3 = 6; therefore, N2 gets 6, and N6 is the index for the "outer" N (because N (6) is equivalent to N6, thus giving N1 = N (N6)). N6 contains 8; thus N1 = N8. Because N8 contains zero, N1 gets this value (0), and 0 is displayed on the message line.

7.8 Errors

The following messages are issued under the CALC command. When an error is detected, the corresponding error message is displayed on the MESSAGE line, followed by a portion of the command where the error was detected.

Message	**Explanation**
Divide by zero error	An attempt was made to divide by zero.
Expression too complex	The expression is too complex; simplify the expression.
Floating point not allowed	Real values (e.g., 5.2) are not allowed.
Illegal exponential operation	Usually occurs when a negative value is used as the right operand. The illegal exponential expression is displayed on the message line. Correct it and rerun CALC.
Illegal expression	The illegal expression is displayed on the message line. Correct it and rerun CALC.
Invalid base character	The base character is not valid, e.g., 1AD.
Invalid numeric constant	The numeric constant is not valid.
MOD by zero error	An attempt was made to take MOD with zero.
Numeric constant too large	A numeric integer constant must be in the range −(2**31) to (2**31)−1.
Unbalanced parenthesis	Either the right or left parenthesis is missing.
Unrecognized identifier	The illegal identifier is displayed on the message line. Correct it and rerun CALC.

CHAPTER 8 ADVANCED AEDIT-86 USAGE

Macros give AEDIT-86 great flexibility and power. Macros can be written to incorporate the CALC command and AEDIT-86 variables. These macros allow you to print dates, directories, use an on-line calculator, and convert letters from uppercase to lowercase and vice versa.

Several macros are included with AEDIT-86 in the file USEFUL.MAC. You must have AEDIT-86 version V2.0 or later to use this macro file. To activate these macros, type the following:

M(ACRO) G(et) USEFUL.MAC<RETURN>

However, it is recommended that you append USEFUL.MAC to your default macro file (AEDIT.MAC), or include the macros you find useful in your default macro file. It is also recommended that the default macro file contain the appropriate configuration command to set the hardware identification for your terminal automatically (e.g., AH=S3E).

The macros in USEFUL.MAC occupy about 1900 bytes of the macro buffer. The default macro size (3072 bytes) is usually sufficient for USEFUL.MAC. However, you may use the invocation control MACROSIZE(*macro_buffer_size*) if you need to increase the macro buffer capacity.

The macros in USEFUL.MAC use the N-variables, N7, N8, and N9, and the S-variable S9 only. All other variables are not affected by USEFUL.MAC macros.

8.1 USEFUL.MAC

The following macros are contained in USEFUL.MAC. They are listed by name with a short descriptive sentence.

Macro Name	Description
<BL>	The space bar may be used in addition to the TAB key to insert tabs.
.	Find the next occurrence of the target string in the FIND buffer.
,	Find the previous occurrence of the target string in the FIND buffer.
L	Convert the character "under" the cursor to a lowercase character.
U	Convert the character "under" the cursor to an uppercase character.
_	Convert the word starting "under" the cursor from uppercase to lowercase letters.
∧	Convert the word starting "under" the cursor from lowercase to uppercase letters.

<CTRL-W>	Move the cursor right to the next word.
<CTRL-Q>	Move the cursor left to the previous word.
]	Set Leftcol one position to the left.
[	Set Leftcol one position to the right.
}	Set Leftcol three positions to the left.
{	Set Leftcol three positions to the right.
+W	If a white space, skip to the right to the next nonwhite character.
+N	If a nonwhite character, skip to the right to the next white space.
+B	If a blank, skip to the right to the next nonblank.
−W	If a white space, skip to the left to the next nonwhite character.
−N	If a nonwhite character, skip to the left to the next white space.
−B	If a blank, skip to the left to the next nonblank.
DT	Insert the date in *mmm dd, yyyy* format (e.g., July 24, 1984)
DM	Insert the date in *dd-mmm-yyyy* format (e.g., 24-Jul-1984)
	Dates are as returned by the UDI DQ$GET$TIME procedure.
<CTRL-B>	Insert the contents of the Block buffer into the text at the cursor position.
PG	Page the text. Header is always written as "Heading."
PP	Page the text. Header is stored in the S9 S-variable.
	PG and PP are meant for paging text. These macros attempt to put headers on empty lines, if possible. They use Tag C and Tag D for internal computations.
CNTR	Center the text in the current line.
DETAB	Convert all tabs from the current position to the end of the file to blanks.
ENTAB	Convert all blanks from the current position to the end of the file to tabs. This macro works very slowly.
SHL	Display the current line number.
SFL	Display the total number of lines in the file.
SFC	Display the total number of characters in the file.

SHP	Display the current position in the line.
‹CTRL-R›	On-line calculator. In INSERT or XCHANGE modes you may enter an arithmetic expression. Press ‹CTRL-R› and the result is displayed at the cursor position.
SMP	Set the indentation and left and right margins according to the values for the paragraph in which the cursor is currently positioned.
NUM	Insert line number prefix on each line in a text file. The macro uses Tag D for internal computations.
0	Set paragraph with indentation 0, left margin 0, right margin 70
2	Set paragraph with indentation 0, left margin 3, right margin 70
3	Set paragraph with indentation 3, left margin 3, right margin 70
4	Set paragraph with indentation 3, left margin 5, right margin 70
5	Set paragraph with indentation 5, left margin 5, right margin 70
6	Set paragraph with indentation 5, left margin 7, right margin 70
7	Set paragraph with indentation 7, left margin 7, right margin 70

8.2 Tips for Writing Macros

The techniques described in this section are very useful within macros. These techniques will help you to understand the macros in USEFUL.MAC and to write your own sophisticated macros.

8.2.1 Send Text to the Message Line

Sending a message to the message line is done using the CALC command with an expression, rather than an assignment statement, as the argument. As stated in Chapter 7, when the argument is an expression, its value is output to the message line even if it is executed within a macro.

Suppose, for example, that N9 contains the current line number, and you want to output this value, with an appropriate title, to the message line. This is done as follows:

C(ALC) N9=*expr*
C(ALC) S9="current line: ‹FETN› 9"
C(ALC) S9

or in macro form:

...CN9= *expr* \ NLCS9="current line: \XN9" \ NLCS9 \ NL \ MM

8.2.2 Simulate "IF *cond* THEN RETURN" (to the caller)

This construct is done by using the "fail" characteristic of the FIND command. Recall that searching an empty string always fails (assuming that SET Go No is in effect), but searching zero times always succeeds, regardless of the operand. The following method may thus be used:

```
C(ALC) N9= cond          \* cond is any logical expression *\
<FETN> 9 F(IND) -<RUBOUT> <ESC>
                         \* Find argument is an empty string *\
...
```

or in macro form:

```
...CN9= cond \NL\XN9F-\RB\BR...
```

Note that *cond* must be a logical expression, so that it will have the value of 0 (false) or −1 (0FFFFFFFFH, true). When it is used as a count, the absolute value, 0 (false) or 1 (true) is used.

8.2.3 Simulate "IF *cond* THEN *statement*"

In this case, *statement* is implemented as a separate macro (named *statement*). This macro will be executed 0 or 1 times depending on the value of *cond*:

```
C(ALC) N9= cond
<FETN> 9 E(XECUTE) statement
```

or in macro form:

```
...CN9= cond \NL\XN9E statement \NL...
```

Note again that *cond* must be a logical value to ensure that N9, when used as a count, is either 0 or 1.

8.2.4 Simulate "IF *cond* THEN *statement1* ELSE *statement2*"

This is based on the example given above:

```
C(ALC) N8=!(N9=cond)     \* "!" is the NEG operator *\
<FETN> 9 E(XECUTE) statement1
<FETN> 8 E(XECUTE) statement2
```

or in macro form:

```
...CN8=!(N9=cond)\NL\XN9E statement1 \XN8E statement2 \NL...
```

Note that *statement*1 may not change N8; if it does, *statement*2 may be executed unintentionally.

8.2.5 Simulate "Advance_While *cond*"

This construct is usually needed for macros like skip to next blank, skip to next word, etc. This is done using the "IF *cond* THEN RETURN" method described above. *cond* in this case must be a logical expression that involves the current cursor location (e.g., ISDEL, CURCH=20H). The method consists of two nested macros. The "low level" macro advances one character and fails when the condition is not met (named Advance_One). The "main" macro executes the first one an infinite number of times and actually terminates when the condition is not met, and continues with the next instruction:

```
Advance_While : ... / E(XECUTE) Advance_One ...

Advance_One : C(ALC) N9=!cond
              <FETN> 9 F(IND) - <RUBOUT> <ESC>
              <RIGHT>
```

or in macro form:

```
MAdvance_While \ BR.../EAdvance_One \ NL...

MAdvance_One \ BRCN9=!cond \ NL \ XN9F- \ RB \ BR \ CR...
```

If, for example, you want to implement skip to the next blank/tab, then *cond* is "CURCH<>20H & CURCH<>09H".

To implement Backward_While (*cond*), the same method is used, but the last command in Advance_One should be <LEFT> (\ CL) instead of <RIGHT> (\ CR).

8.3 Examples

The examples included in this section are macros from USEFUL.MAC. They are explained in greater detail to show the usage of the above techniques.

8.3.1 Example 1

The following set of macros converts single letters or words from uppercase to lowercase or vice versa. It executes nested macros (e.g., macro L calls macro U2L), uses the CALC command (e.g., C(ALC) n8=(n9=lowch) and read-only variables (e.g., lowch, curch, upch), calls the SET command (S(ET) R(adix) A) and the fetch function (e.g., <FETN> 8).

```
ML \ BReu2l \ NL \ CR \ XN8eLU11 \ BR \ MM;      \* letter to lower case *\
MU \ BRel2u \ NL \ CR \ XN8eLU11 \ BR \ MM;      \* letter to upper case *\
M_ \ BRe+W \ NL/el12 \ NLe+W \ BR \ MM;          \* word to lower case *\
M^ \ BRe+W \ NL/eu12 \ NLe+W \ BR \ MM;          \* word to upper case *\
MU2L \ BRCN8=(N9=lowch)<>curch \ NL \ MM;
ML2U \ BRCN8=(N9=upch )<>curch \ NL \ MM;
MLU11 \ BR \ CLsrax \ XN9 \ BR \ MM;
ML12 \ BRCN7=iswhte|eof \ NL \ XN7f- \ RB \ BRl \ MM;
MU12 \ BRCN7=iswhte|eof \ NL \ XN7f- \ RB \ BRu \ MM;
```

Note the use of the following techniques:

- Macro L executes LU11 using the IF *cond* THEN *statement*
- Macro L12 simulates Advance_While the current character is not a white space. In this case, it also converts the character to lowercase.
- Macro _ (underscore) uses +W to skip to the next nonwhite space character.

8.3.2 Example 2

Macro <CTRL-W> moves the cursor one word to the right. A word (in this case) is defined as a sequence of characters enclosed by delimiters. Delimiters are defined as white spaces or the user defined delimiters (listed under SET E_delimiter). The technique IF *cond* THEN RETURN is used here. In the nested macros, N9 defines whether or not the cursor is on a delimiter. If the value fetched by <FETN> 9 is 0 (false), the FIND succeeds and the macro is repeated. If the value fetched by <FETN> 9 is 1 (true), the FIND fails and execution returns to the calling macro.

```
M\017\BR0f-\RB\BR/e\0171\NL/e\0172\NL\MM;  \* word right macro *\
M\0171\BRcn9=iswhte\NL\XN9f\BR\CR\MM;
M\0172\BRcn9=!isdel\NL\XN9f\BR\CR\MM;
```

Macro <CTRL-Q> differs from macro <CTRL-W> only in that it moves the cursor one word to the left.

```
M\011\BR0f-\RB\BR\CL/e\0111\NL/e\0112\NL\CR\MM; \* word left macro *\
M\0112\BRcn9=iswhte\NL\XN9f\BR\CL\MM;
M\0111\BRcn9=!isdel\NL\XN9f\BR\CL\MM;
```

Note that the macros are optimized to some extent. A find empty string is issued first; therefore, a future FIND command uses <ESC> as an argument, not the sequence -<RUBOUT> <ESC>.

8.3.3 Example 3

Macro CNTR centers the text on the line. This macro strips all blanks from the left side, then all blanks from the right side. It calculates the number of blanks to be added and adds the blanks from the left margin to the first character so that the text is centered on the line. Skipping blanks is done with the +B and the −B macros. The number of blanks to be added is calculated using the read-only variables RMARGN, LMARGN, and COL. +B and −B use Advance_While macro techniques.

```
MCNTR\BRjp0\NLe+b\NL\XXjp254\NLi\BR\CLe-b\NL\CR
\XA cn9=(rmargn+1+lmargn-col)/2\NLcn9=-n9*(n9>0)\NLjp0\NLi\BRb
\CLb\XF\XN9g\NL\NL\MM;
```

```
M+B\BR/e+B1\BR\MM;
M+b1\BRcn9=!(curch==20H)\NL\XN9f-\RB\BR\CR\MM;
```

```
M-B\BR/e-B1\BR\MM;
M-b1\BRcn9=!(curch==20H)\NL\XN9f-\RB\BR\CL\MM;
```

CHAPTER 9
CONFIGURATION COMMANDS

9.1 Introduction

AEDIT-86 is designed to run within several environments and with various terminals. In some cases—for example, Series IV AEDIT-86 is able to identify the host environment. In other cases, you should specify the characteristics of your particular environment or terminal.

The characteristics of your environment or terminal are specified with configuration commands. The parameters and control sequences that must be specified are listed in Table 9-1.

Configuration commands must be given in a macro file. It is recommended that the required commands for your environment or terminal be included in your default macro file. A configuration command must be terminated with a semicolon (;) or a carriage return. For the Series III, Series IV and ANSI/VT100 family of terminals, only the appropriate hardware identification configuration command, AH=string, is required. Appendix F lists macro files for several non-Intel terminals. If your terminal is not included in the list of tested terminals, please refer to your terminal user manual for the terminal characteristics.

AEDIT-86 requires that the terminal meet the following conditions:

- ASCII codes 20H–7EH display some symbol that requires one column space. Carriage return (0DH) and linefeed (0AH) perform their usual functions.
- The following cursor functions have cursor key input codes and CRT cursor output codes: left, right, up, down, home and carriage return. Output codes such as clear screen, clear rest of screen, clear line, clear rest of line, and direct cursor addressing are desirable for faster screen plotting, but not required. The codes, shown in Table 9-1, can be changed with the configuration commands.
- The CRT accepts a blankout code that blanks out the former contents of the screen location to which it is output. The default, 20H, can be changed with the configuration commands.
- AEDIT-86 automatically generates a linefeed each time a carriage return is entered. Your terminal should not transmit a linefeed with a carriage return. In some terminals, this feature can be switched on and off.

Table 9-1 lists the configuration commands and their meanings. These commands are divided into three groups:

- Terminal attributes and generals
- Input codes—specify codes sent from the keyboard to the terminal (i.e., AEDIT-86)
- Output codes—specify codes to be sent from the terminal (i.e., AEDIT-86) to the display.

A configuration command may be used to set a value to a specific feature or to indicate that the feature is not available. To indicate that the feature is not available, the command is specified without an associated value, e.g., AFER=; .

Table 9-1. Configuration Commands

Command	Meaning
Terminal Attributes and Generals	
AH=*string*	**Hardware Identification**
string stands for one of the following values. Each of the specified values implies a set of configuration commands.	
‹ null ›	Equivalent to specifying the minimal default set, which is specified with AH=;. The default set is not sufficient for interactive use and must be completed with other explicit configuration commands.
S3	Series III systems. Equivalent to specifying all configuration commands with the Series III default values.
S3E	Equivalent to S3 with the following changes: • New console output functions, including— -Various clear text functions -Direct cursor addressing -Local scrolling • Fast-Block-Move command for data to the CRT
S3ET	Equivalent to S3E without the Fast-Block-Move command for data to the CRT. S3ET is used when the Series IIIE is used as a terminal rather than as a host.
S4	Series IV systems. Equivalent to specifying all configuration commands with the Series IV default values.
ANSI	Equivalent to specifying all the configuration commands with the default values according to ANSI X3.64 (1977). The ANSI cursor addressing codes, as defined in this standard, are supported.
VT100	Equivalent to specifying all the configuration commands with the default values for the DEC VT100 family of terminals. This includes VT100 ANSI cursor addressing codes.
AV=n (5:66)	Sets the number of rows in the display.
AW=T or F	True if the terminal wraps when a character is printed in the last physical column.
AS=T or F	True to display the busy/waiting indicator.
AT=T or F	True if type-ahead is to be done by AEDIT-86. Must be False for iRMX-based system.
Input codes	
Escape keys:	
AB=*hhhh*	Sets ‹ ESC ›
AFCC=*h*	Sets ‹ CTRL-C › synonym
Cursor move keys:	
AFCL=*hhhh*	Sets ‹ LEFT ›
AFCR=*hhhh*	Sets ‹ RIGHT ›
AFCU=*hhhh*	Sets ‹ UP ›
AFCD=*hhhh*	Sets ‹ DOWN ›
AFCH=*hhhh*	Sets ‹ HOME ›

Table 9-1. Configuration Commands (Cont'd.)

Command	Meaning
Delete keys:	
AR=*hhhh*	Sets <RUBOUT>
AFXF=*hhhh*	Sets Delete Character- <DELCH>
AFXX=*hhhh*	Sets Delete Left- <DELL>
AFXA=*hhhh*	Sets Delete Right- <DELR>
AFXZ=*hhhh*	Sets Delete Line- <DELLI>
AFXU=*hhhh*	Sets UNDO- <UNDO>
Prefix keys:	
AFXE=*hhhh*	Sets Macro Exec key- <MEXEC>
AFXH=*hhhh*	Sets HEX character- <HEX>
AFXN=*hhhh*	Sets Fetch Numeric- <FETN>
AFXS=*hhhh*	Sets Fetch String- <FETS>
Others:	
AFIG=*h*	Sets character(s) to be ignored. This specification is needed on terminals (such as the Hazeltine 1510) that have multiple character key codes for UP and DOWN. AFIG should be set to the lead-in (tilde), and UP and DOWN should be set to the second letter of the cursor up or down key code. This avoids problems caused by the lack of a type-ahead buffer.
Output Codes	
Blank:	
AFBK=*h*	Blankout character <BLANK> on most terminals
Cursor moves:	
AFMB=*hhhh*	Moves cursor to start of line
AFML=*hhhh*	Moves cursor left
AFMR=*hhhh*	Moves cursor right
AFMU=*hhhh*	Moves cursor up
AFMD=*hhhh*	Moves cursor down
AFMH=*hhhh*	Moves cursor home
Erase:	
AFES=*hhhh*	Erases entire screen
AFER=*hhhh*	Erases rest of screen
AFEK=*hhhh*	Erases entire line
AFEL=*hhhh*	Erases rest of line
Cursor addressing:	
AFAC=*hhhh*	Addresses cursor lead-in. When used, code will be followed by column number (0 to *max_col_value*) and row number (0 to *max_row-_value*).
AO=*h*	Offset to add to both row and column number with address cursor commands.
AX=T or F	True if X (column) precedes Y (row) in address cursor command.

Table 9-1. Configuration Commands (Cont'd.)

Command	Meaning
Delete/insert:	
AFIL=*hhhh*	Insert line code. Used in line 0 for reverse scrolling.
AFDL=*hhhh*	Delete line code. Used to speed up display on the Hazeltine 1510 and similar terminals.
Reverse video:	
AFRV=*hhhh*	Start reverse video character(s). Used on the PROMPT line display.
AFNV=*hhhh*	Return to normal video characters. Used to restore the display after a reverse video line.
AI=T or F	True if the CRT has invisible attributes, or False if the attribute occupies a position on the screen.
AC=T or F	True if the CRT has attributes per character, or False if the attributes are per field.
Initialization/termination:	
AFST=*hhhh...*	Start-sequence. This sequence is output to the terminal when AEDIT-86 encounters it. Can be used to initialize the terminal, e.g., to initialize a scrolling region.
AFEN=*hhhh...*	End-sequence. Output to the terminal when AEDIT-86 exists. Can be used to unset values that have been set by AFST.

NOTES:

n(*n1*:*n2*)	an integer in the inclusive range, *n1* to *n2*.
h	a 1-byte hex number.
hhhh	a 1- to 4-byte hexadecimal number.
hhhh...	the same as hhhh, but the length may be up to 40 bytes.
T	*T* or *t* indicates true.
F	*F* or *f* indicates false.
string	a 0- to 60-character string.

9.1.1 Configuration Command Notes

AT This feature must be off (AT=F) for iRMX-based systems due to current iRMX internal requirements. Setting AT=F directs AEDIT-86 not to support type-ahead. Because the iRMX operating system performs type-ahead, setting this function off does not affect the user.

When AT is off (AT=F), the AFCC function is not fully functional. The synonym works only for synchronous operations (e.g., to terminate INSERT mode), but not for asynchronous operations (e.g., exit from a loop within a macro).

AB The natural choice is ESC (1BH). However, this value cannot be used if other terminal input functions begin with ESC (e.g., VT10).

AFCC AFCC is not fully functional if AT is off (AT=F). See notes on AT above.

9.2 Configuration Values

The hardware identification command AH may have several different values, e.g., a null string, the string S3E, the string S4. An AH command value implies a complete set of values for the entire configuration command set. Table 9-2 contains the default configuration commands with values for various terminals or systems.

Table 9-2. Configuration Values

Command	AH=; Default	AH=S3 (S III)	AH=S3E (S IIIE)	AH=S4 (S IV)	AH=ANSI AH=VT100	Meaning
Terminal Attributes and Generals						
AV=	24	25	25	25	24	Screen length
AW=	T	F	T	F	F	Terminal wraps?
AS=	T	T	T	T	T	Display busy/waiting indicator
AT=	T	T	T	T	T	Type-ahead
Input codes						
Escape keys:						
AB=	1B	1B	1B	1B	1B4F53(2)	Sets ‹ESC›
AFCC=	03	03	03	03	03	Sets ‹CTRL-C›
Cursor move keys:						
AFCL=	—(1)	1F	1F	89	1B5B44	Sets ‹LEFT›
AFCR=	—	14	14	8A	1B5B43	Sets ‹RIGHT›
AFCU=	—	1E	1E	87	1B5B41	Sets ‹UP›
AFCD=	—	1C	1C	88	1B5B42	Sets ‹DOWN›
AFCH=	—	1D	1D	81	1B4F50(4)	Sets ‹HOME›
Delete keys:						
AR=	7F	7F	7F	7F	7F	Sets ‹RUBOUT›
AFXF=	06	06	06	80	06	Sets ‹DELCH›
AFXX=	18	18	18	18	18	Sets ‹DELL›
AFXA=	01	01	01	01	01	Sets ‹DELR›
AFXZ=	1A	1A	1A	82	1A	Sets ‹DELLI›
AFXU=	15	15	15	15	15	Sets ‹UNDO›
Prefix keys:						
AFXE=	05	05	05	05	05	Sets ‹MEXEC›
AFXH=	08	08	08	08	08	Sets ‹HEX›
AFXN=	0E	0E	0E	0E	0E	Sets ‹FETN›
AFXS=	13	13	13	13	16(3)	Sets ‹FETS›
Others:						
AFIG=	—	—	—	—	—	Ignore character
Output codes						
Blank:						
AFBK=	20	20	20	20	20	Blankout character
Cursor moves:						
AFMB=	0D	0D	0D	0D	0D	Carriage return
AFML=	—	1F	1F	08	1B5B44	Cursor left
AFMR=	—	14	14	1B43	1B5B43	Cursor right
AFMU=	—	1E	1E	1B41	1B5B41	Cursor up
AFMD=	—	1C	1C	1B42	1B5B42	Cursor down
AFMH=	—	1D	1D	1B48	1B5B48	Cursor home
Erase:						
AFES=	—	1B45	1B45	1B45	1B5B324A	Entire screen
AFER=	—	1B4A	1B53	1B4A	1B5B4A	Rest of screen
AFEK=	—	1B4B	1B4B	0D1B4B	1B5B324B	Entire line
AFEL=	—	—	1B52	1B4B	1B5B4B	Rest of line

Table 9-2. Configuration Values (Cont'd.)

Command	AH=; Default	AH=S3 (S III)	AH=S3E (S IIIE)	AH=S4 (S IV)	AH=ANSI AH=VT100	Meaning
Cursor addressing:						
AFAC=	—	—	1B59	1B59	1Bx; y48[5]	Address lead in
AO=	—	—	20	20	—	Row/column offset
AX=	—	—	F	F	—	X (col) before Y (row)
Delete/insert:						
AFIL=	—	—	1B57603F	—	1B5B4C[6]	Insert line
AFDL=	—	—	1B573F60	—	1B5B4D[6]	Delete line
Reverse video:						
AFRV=	—	—	1B4C90	1B4C50	1B5B376D	Reverse video
AFNV=	—	—	1B4C80	1B4C40	1B5B6D	Normal video
AI=	—	—	F	F	T	Invisible attributes?
AC=	—	—	F	F	T	Character attributes
Initialization/ termination:						
AFST=	—	—	—	—	—	Start-sequence
AFEN=	—	—	—	—	—	End-sequence

NOTES:

1. — means the feature is either unavailable or meaningless.
2. Because 1B is used as a prefix for input sequences on ANSI terminals, it may not be used as < ESC >. The choice of the FP4 key (for AH=VT100) or 04H (for AH=ANSI) is arbitrary and may be changed.
3. < CTRL-S > may cause problems with the VT100 and other terminals. Choose another key (e.g., < CTRL-V >).
4. In the absence of a "natural" < HOME > key, the choice of the FP1 key (for AH=VT100) or 0CH (for AH=ANSI) is arbitrary and may be changed.
5. The ANSI escape sequence for cursor addressing is hard-coded in AEDIT-86. The table shows the format only. This format cannot be coded using AFAC.
6. Insert/Delete line functions, although available on the VT100, are disabled because of poor performance.

9.3 Delay Codes

Some CRTs are too slow with respect to some AEDIT-86 output functions. To enable a smooth operation of AEDIT-86 with these CRT types, AEDIT-86 should be informed how long it has to wait before it may issue a new output operation.

Delay codes for the various output functions can be specified by configuration commands of the form AD*xx*=*hhhh*, where *xx* is the function code in the corresponding AF*xx* code, and *n* is a decimal number specifying the delay in milliseconds. An AD*xx* value may be specified for every output function for which a corresponding AFxx value may be applied. For example, ADDL=30 defines a delay of 30 milliseconds for the function AFDL (delete line).

The default for all delay codes is no delay at all.

9.4 Determining the Configuration Values

The default values are given in Table 9-2.

The following list describes the order in which AEDIT-86 processes configuration-command-related information:

1. Sets the default value for each configuration command.
2. Checks to see if the system is a S III. If the system-id is SERIES III, then sets the configuration commands as defined under S III in Table 9-2.
3. Checks to see if the system is S IV. If so, then sets the configuration commands as defined under S IV in Table 9-2.
4. Processes the macro file (which may contain configuration commands). For every legal command, overwrites the previous value with the one newly specified.

   When the above process is finished, the configuration set is checked automatically. The following configuration commands must have a defined value:

   AFMB, AFMH, AFMU, AFMD, AFML, AFMR, and AFBK.

During AEDIT-86 operation, new macro files can be loaded using the MACRO Get command. After each addition, step 4 above is repeated, but the check for the minimal set of defined configuration commands is not carried out.

NOTE

AEDIT performs no consistency checks on the configuration commands. In particular, there is no check as to whether two input codes have the same hex value or one is the prefix of the other. In such a case, the result is undefined.

Error. "insufficient configuration commands" is displayed and the AEDIT-86 session is terminated with the error if one or more of the configuration commands AFMB, AFMH, AFMU, AFMD, AFML, AFMR, or AFBK does not have a defined value.

APPENDIX A
AEDIT-86 COMMAND SUMMARY

This appendix lists the AEDIT-86 commands, their formats and functions. In the alphabetic list, <CTRL> represents the CONTROL key. Angle brackets (< >) indicate a key configured for a function.

A.1 Function Keys

The following commands are executed by pressing the specifically configured key or by typing the indicated <CTRL> commands.

Command	Function
<LEFT>	Moves cursor left, left arrow.
<RIGHT>	Moves cursor right, right arrow.
<UP>	Moves cursor up, up arrow.
<DOWN>	Moves cursor down, down arrow.
<HOME>	Allows fast cursor movement or permits entry into reedit mode.
<RUBOUT>	Deletes character to left of cursor at main command level or INSERT mode. In XCHANGE mode, <RUBOUT> exchanges the new character to the left of the cursor with the original character.
<DELCH>	Deletes the character "under" the cursor (usually configured to <CTRL-F>).
<DELLI>	Deletes the entire line on which the cursor is positioned (usually configured to <CTRL-Z>).
<DELL>	Deletes all characters to the left of the cursor (usually configured to <CTRL-X>).
<DELR>	Deletes all characters to the right of the cursor (usually configured to <CTRL-A>).
<UNDO>	At the cursor position, restores the characters deleted by the last <DELL> , <DELR> , or <DELLI> command (usually configured to <CTRL-U>).
<ESC>	Terminates commands and sends the entire string, or exits the mode and returns editor to main command level.
<CTRL-C>	Aborts command in progress and returns editor to main command level.
<RETURN>	Moves cursor to start of next line.
<TAB>	Rotates the menu prompt line to display the next line of commands or, in INSERT or XCHANGE mode, inserts tab or equivalent number of blanks.

< HEX >	Inserts a character in the text as its ASCII value (usually configured to < CTRL-H >). < HEX > should be followed by two digits that are interpreted as the hexadecimal value.
< FETN >	Fetches a global numeric variable (usually configured to < CTRL-N >). < FETN > should be followed by a numeric value from 0–9.
< FETS >	Fetches a global string variable (usually configured to < CTRL-S >). < FETS > should be followed by a numeric value from 0–9, or a second letter from one of the read-only string variables.
< MEXEC >	Executes a macro called by a single-character name (usually configured to < CTRL-E >). < MEXEC > should be followed by a valid one-character macro name.

A.2 AEDIT-86 Editing Commands

The following is an alphabetic list of the AEDIT-86 editing commands, their formats and functions.

Command	Format	Function
AGAIN	[*count*]A	Repeats the last command or subcommand.
BLOCK	B or D	Delimits a section of text that can then be deleted, moved, or copied.
Buffer	B	Copies text to the Block buffer.
Delete	D	Deletes the delimited section and moves it unchanged to the Block buffer.
Find	F	Same as in main command level.
-find	-	Same as in main command level.
Jump	J	Same as in main command level.
Put	P	Copies the delimited section of text to a specified output file.
DELETE	B or D	Delimits section of text that can then be deleted, moved, or copied (same as in BLOCK command).
FIND	[*count*] F	Searches forward from current cursor position for string. Moves cursor if found.
-FIND	[*count*] -	Searches backward from current cursor position for string. Moves cursor if found.

GET	[*count*] G	Retrieves contents of Block buffer or external file; places contents at current cursor position. Count must be a finite number.
HEX	[*count*] H	Hex command
Input	I	Inserts the ASCII equivalent of hexadecimal values in text.
Output	O	Displays hexadecimal values of ASCII characters in message line.
INSERT	[/] I	Begins INSERT mode; inserts text at cursor position.
JUMP	J	Moves cursor to a specified location in text.
A_tag	A	Moves cursor to tag A.
B_tag	B	Moves cursor to tag B.
C_tag	C	Moves cursor to tag C.
D_tag	D	Moves cursor to tag D.
Start	S	Moves cursor to the start of the file.
End	E	Moves cursor to the end of the file.
Line	L	Moves cursor to the start of the designated line.
Position	P	Moves cursor to the designated position in the current line.
KILL_WND	K	Deletes the secondary window and extends the current window. The cursor remains at its current position.
OTHER	O	Switches between main and OTHER files.
PARAGRAPH	[*count*] P	Reformats the paragraph using the values for indentation, left and right margins set with the SET Margin command.
Fill	F	The paragraph is reformatted with no right-justification
Justify	J	The paragraph is reformatted with right-justification.
QUIT	Q	Ends, updates, restarts, etc., the editing, depending on the subcommand.

Abort	A	Returns to operating system; all changes are lost.
Exit	E	Returns to operating system; the file is updated.
Init	I	Restarts editing; initializes new file without returning to operating system.
Update	U	Updates file without returning to operating system.
Write	W	Writes file to the output file specified without returning to operating system.
REPLACE	[*count*] R	Searches forward for target string; replaces it with new string if found.
?REPLACE	[*count*] ?	Conditional REPLACE command.
SET	S	Sets several AEDIT-86 features.
Autonl	A	While in INSERT mode, inserts carriage return in text automatically when line is full (default=no).
Bak_file	B	Creates a backup file of the file being currently edited when QUIT Update or QUIT Exit is executed (default=yes).
Case	C	Tells editor to consider case of strings during (-)FIND and (?)REPLACE commands (default=no).
Display	D	Displays any movements in or changes to the text during macro execution (default=no).
E_delimit	E	Defines the token delimiters (default = ! " # % & ' ( ) * + , − . / : ; < = > ? @ [\ ] ^ ` { | } ~).
Go	G	Continues macro execution even if a (-)FIND/(?)REPLACE command in the macro fails (default=no).
Highbit	H	Displays all text characters with hexadecimal values over 7FH as is instead of ? (default=no).
Indent	I	Indents inserted/exchanged text automatically (default=no).

K_token	K	A string in the text needs to be a token string to be found (default=no).
Leftcol	L	Allows you to view lines over 80 characters long (default=0).
Margin	M	Sets indentation, and left and right margins used by the PARAGRAPH and SET Autonl commands (default: indent=4, left=0, right=76).
Notab	N	Inserts blanks in place of tabs in INSERT or XCHANGE mode (default=no).
Radix	R	Sets the radix by which a numeric variable is output by <FETN> in INSERT or XCHANGE mode (default = Decimal).
Alpha	A	
Binary	B	
Decimal	D	
Hex	H	
Octal	O	
Showfind	S	Lists *target_string* lines of multiple search commands (default=no).
Tabs	T	Sets tab positions (default = 4).
Viewrow	V	Sets the row to which text is moved relative to the screen on VIEW command (default= row 5).
TAG	T	Specifies locations in a file; used with the JUMP command.
A_tag	A	
B_tag	B	
C_tag	C	
D_tag	D	
VIEW	V	Rewrites (moves) text on the screen leaving cursor in viewrow (default=row 5).

WINDOW	W	Splits the text area of the screen in two, enabling the user to look at two different parts of the same file or two different files.
XCHANGE	X	Enters XCHANGE mode; replaces characters on a one-for-one basis.

A.3 AEDIT-86 Advanced Commands

The following is an alphabetic list of the AEDIT-86 advanced commands, their formats and functions.

Command	Format	Function
CALC	C	Provides computing capabilities.
EXECUTE	[*count*] E	Executes specified macro.
MACRO	M	Allows you to manipulate macros.
Create	C	Creates macros interactively by accumulating a sequence of keystrokes.
Get	G	Retrieves macros from an external file or from the current text buffer.
Insert	I	Inserts subsequent input in text in macro form.
List	L	Lists the names of all currently defined macros on the message line.
Save	S	Translates macros to macro form and inserts the definition at the current position in the text.

APPENDIX B AEDIT-86 ERROR MESSAGES

This appendix lists the error messages reported by AEDIT-86 when a problem is encountered in the invocation line, editing command, CALC command, or macro file.

B.1 Invocation Errors

AEDIT-86 issues a message when an error occurs in the invocation line. AEDIT-86 displays the sign-on message followed by the error message, and control returns to the operating system.

Invocation errors have the following form:

```
***ERROR:   illegal invocation, NEAR:   token
```

or

```
***ERROR:   message
```

where:

token	is an invocation command item.
message	is one of the following error messages.

Message	Explanation
Conflicting controls	An illegal combination of VIEWONLY and FORWARDONLY controls is used.
Insufficient configuration commands	The interactive session has been initialized; however, required configuration commands of the type AFM*x* or AFBK are undefined.
Insufficient memory	AEDIT-86 does not have a large enough RAM partition.
Macro buffer too large	The macro buffer size specified is too large, and the buffer size remaining is insufficient for the text to be edited.
Macro buffer too small	The macro buffer size specified is less than the minimum allowed.

B.2 Editing Command Errors

Message	Explanation
bad indent margin	Attempt to set indent margin out of the legal range. Editor returns to main command level.
bad left margin	Attempt to set left margin out of the legal range. Editor returns to main command level.

bad margins	Attempt to set margins out of the legal range. Editor returns to main command level.
bad right margin	Attempt to set right margin out of the legal range. Editor returns to main command level.
bad tabs	Attempt to set bad tabs; e.g., 4,2 is illegal since the second value is less than the first. Editor returns to main command level.
bad Leftcol	Attempt to set bad left column, Leftcol accepts any number from 0 to 176. Editor returns to main command level.
bad Viewrow	Attempt to set bad viewrow. This value must be between 0 and (text size — 1). Editor returns to main command level.
block buffer too large for SB	Attempt to specify ‹FETS› B or SB when the current block buffer is greater than 60 characters.
cannot delete more than 32	Attempt to use count greater than 32 with ‹DELCH› command. Editor returns to main command level.
illegal command	Attempt to enter illegal and/or unknown command. Editor ignores command.
illegal invocation	Attempt to invoke AEDIT-86 with an illegal invocation line, or an illegal invocation under QUIT Init.
insufficient terminal capabilities	Attempt to set windowing on a terminal that does not have AFEK or AFEL output functions. Editor returns to main command level.
invalid hex value	Attempt to input an invalid hexadecimal value. Editor returns to main command level.
invalid variable name	Attempt to input an invalid value for ‹FETS› or ‹FETN›.
macro creation is forbidden while executing a macro	Attempt was made to define a macro while a macro was being executed.
macro nesting too deep	Nesting level for macros exceeded. A maximum of eight levels is allowed.
no more room for macros	Macro buffer is full.
no such macro	Attempt to execute macro that does not exist. Editor returns to main command level.
not found: "*target string*"	Target string not found. Editor returns to main command level.
some text lost	In FORWARDONLY, the file is larger than the allocated memory.

text does not fit	Attempt to edit a file that is too large under the FORDWARDONLY control. Editor returns to main command level.
filename (*error message supplied by operating system*)	An error occurs during a QUIT Exit, QUIT Init, QUIT Update, GET, or BLOCK Put command. Editor returns to main command level.
Xchange limit is 100	Attempt to exchange over 100 characters without restarting XCHANGE mode. Editor remains in XCHANGE mode.
window too small	Attempt to split screen with one window size less than five lines.

B.3 CALC Command Errors

The following messages are issued only under the CALC command. When an error is detected, the corresponding error message is displayed on the MESSAGE line, followed by a portion of the command where the error was detected.

Message	Explanation
Divide by zero error	An attempt was made to divide by zero.
Expression too complex	The expression is too complex; simplify the expression.
Floating point not allowed	Real values (e.g., 5.2) are not allowed.
Illegal exponential operation	Usually occurs when a negative value is used as the right operand. The illegal exponential expression is displayed on the message line. Correct it and rerun CALC.
Illegal expression	The illegal expression is displayed on the message line, correct it and rerun CALC.
Invalid base character	The base character is not valid, e.g., 1AD.
Invalid numeric constant	The numeric constant is not valid.
MOD by zero error	An attempt was made to take MOD with zero.
Numeric constant too large	A numeric integer constant must be in the range −(2**31) to +(2**31)−1.
Unbalanced parenthesis	Either the right or left parenthesis is missing.
Unrecognized identifier	The illegal identifier is displayed on the message line. Correct it and rerun CALC.

B.4 Macro File Errors

If any error is found in a macro file, one of the following messages is printed. Macro file processing continues.

Macro errors have the following form:

```
Error in line nnn: <message>
```

where

nnn	is the line number containing the error in the macro file.
‹message›	is one of the following error messages.

Message	Explanation
bad \ code	Backslash (\ ) is not followed by a valid value.
bad AC value	Illegal AC configuration command.
bad AF type	Illegal AF configuration command.
bad AH value	Illegal value for the AH configuration command.
bad AI value	Illegal AI configuration command.
bad AS value	Illegal AS configuration command.
bad AT value	Illegal AT configuration command.
bad AV value	Illegal value for the AV configuration command.
bad AW value	Illegal AW configuration command.
bad AX value	Illegal value for the AX configuration command.
bad command	Macro file contains a bad command—unknown control code or character.
bad hex value	Configuration command contains bad hex value, e.g., 3G.
missing '='	A configuration command is missing an equal sign.
missing ';'	SET command is not terminated with a semicolon or carriage return.
no macro name	Macro definition does not include macro name.
no more room for macros	Attempt to create a macro when macro buffer is full. Macro definition is terminated.

APPENDIX C
USING AEDIT-86 ON THE SERIES III

AEDIT-86 will run on the Series III Microcomputer Development System under the ISIS/RUN operating system.

C.1 Invocation Command

The invocation line is as described in Chapter 4. If you are not in the RUN subsystem, the invocation must be preceded by RUN.

If you are starting the tutorial in Chapter 1, type the following:

```
RUN AEDIT <RETURN>
```

C.2 Terminals Used with the Series III System

The Series III system may be connected to various terminals:

- Integrated Series III terminal
- Integrated Series IIIE terminal (with the 511IOC firmware)
- Other terminals connected to the Series III via a serial channel

Descriptions of the Series III and Series IIIE follow in this appendix. See Appendix F for working with other terminals.

C.3 Integrated Series III Terminal

C.3.1 Configuration Commands

The hardware identification value AH is S3 for the integrated Series III terminal. This is the default hardware identification for AEDIT-86 if the system-id, as returned by UDI, is Series III. Configuration commands for the Series III terminal are presented in Table 9-2.

C.3.2 Scrolling

Scrolling at the main command level causes the message and prompt lines to disappear. When scrolling is complete, the message and prompt lines are rewritten as soon as a command (other than a cursor movement command) is typed. The integrated Series III terminal has no reverse scrolling.

C.4 Integrated Series IIIE Terminal

The integrated Series IIIE terminal is an upgrade of the integrated Series III terminal. An integrated Series III terminal may be upgraded to Series IIIE by installing the 511IOC firmware. The integrated Series IIIE terminal has the following additional features:

- Direct cursor addressing
- Additional output function codes (e.g., delete line, insert line)
- Fast block movement to the CRT

AEDIT-86 executes much faster on the integrated Series IIIE terminal than on the regular integrated Series III terminal.

C.4.1 Configuration Commands

To use Series III with the integrated Series IIIE terminal, use AH=S3E configuration command. If the integrated Series IIIE terminal is used as a terminal for another host (via asynchronous communication), use AH=S3ET. In this case, the fast block movement feature is not exploited. Configuration commands are presented in Table 9-2.

C.4.2 Using Serial Channels with the Integrated Series IIIE Terminal

If you are using AEDIT-86 with the integrated Series IIIE terminal and you have devices attached to either of the serial channels, your terminal could "freeze" if a device sends data sequences to the terminal. Disconnecting such devices beforehand is recommended; however, this problem, when it occurs, is resolved by pressing the system Interrupt 7 key.

C.5 Work File

The predefined file :WORK: must be properly assigned when AEDIT-86 is invoked.

If you are using ISIS-II/RUN, assign :WORK: to a write-enabled device with the RUN WORK command.

If you are using ISIS-III/RUN, :WORK: is always assigned to :F9:. Therefore, assign :F9: to a write-enabled device/directory.

C.6 Related Publications

For more information on the Series III Microcomputer Development System, see the following manuals:

- *Intellec® Series III Microcomputer Development System Overview*, order number 121575
- *Intellec® Series III Microcomputer Development System Programmer's Reference Manual*, order number 121618
- *Intellec® Series II CRT and Keyboard Interface Manual*, order number 122029

APPENDIX D
USING AEDIT-86 ON THE SERIES IV

AEDIT-86 runs on the iNDX operating system on the Intellec Series IV Microcomputer Development System.

D.1 Invocation Command

The invocation line is as described in Chapter 4.

If you are starting the tutorial in Chapter 1, type the following:

```
AEDIT <RETURN>
```

D.2 Keyboard

The Series IV keyboard has soft keys. Therefore, all commands may be invoked either with the initial letter of the command or with the corresponding soft key.

CHAR DELETE	The CHAR DELETE (delete character) key deletes the character "under" the cursor.
CLEAR LINE	The CLEAR LINE key deletes the entire line on which the cursor is positioned.

Most keys have a built-in automatic repeat function. To enter multiple characters, simply continue pressing the same key. The < HOME > and < ESC > keys do not repeat automatically.

D.2.1 Soft Keys

Eight function keys labeled F0–F7 are located at the top of the keyboard. They correspond to the menu prompt line displayed on the bottom of the screen. These keys are called soft keys because their meaning changes depending on which menu prompt line is displayed. Pressing the soft key that corresponds to a menu selection displayed on the screen executes that command. Press the eighth soft key or < TAB > to display the next set of options.

The function keys may be also used as single character macros under INSERT or XCHANGE mode. To define a macro associated with < F1 >, for example, you should type—

```
M(ACRO) C(REATE) <F1> <RETURN>
```

The hex code of < F1 > appears on the screen when the < F1 > key is pressed. The macro is activated simply by pressing < F1 > while in INSERT or XCHANGE mode.

Error. "illegal command" is displayed if a soft key that has no corresponding option is pressed. This is because some prompt lines have fewer than eight options; therefore, not all the soft keys are used.

D.3 Message Line

The message line on the Series IV is 60 columns wide instead of 80. The last 20 columns are reserved for the Series IV system.

D.4 Scrolling

The Series IV has no reverse scrolling. Reverse scrolling is implemented as the VIEW command, and the screen is rewritten.

D.5 Printing and Nonprinting Characters

Except for the ESC, NUL, BELL, and backspace, all control characters are displayed on the screen as is and not as a question mark (?).

D.6 Configuration Commands

The hardware identification value AH is S4 for the Series IV. Configuration commands are presented in Table 9-2.

D.7 Default Macro File

If AEDIT-86 is not located in the current directory, you may have problems activating the default macro file.

The name of the default macro file is AEDIT.MAC, which is assumed to be in the current directory. If it is not, you must explicitly specify the MACRO(*filename*) control.

D.8 Using Series IV with Remote Terminals

When a remote terminal is connected to the Series IV, the configuration file is different.

The communication program (STTY) takes care of most of the terminal specific input/output codes. The only items that must be in the configuration file are—

- The terminal length (if not 25 lines)
- Codes for ‹DELCH› and ‹DELLI› because they almost always differ from the Series IV CHAR DELETE and CLEAR LINE codes

A sample configuration file for a terminal with 24 lines is—

```
AH=S4;
AV=24;
AFXF=06; AFXZ=1A;
```

NOTE

Do not use the regular configuration file for that terminal. For example, if you attach a Zentec terminal to your Series IV, use the configuration command given above and not ZENTEC.MAC.

D.9 Work File

The predefined file :WORK: must be properly assigned when AEDIT-86 is invoked. Under Series IV, :WORK: is automatically assigned to your home directory when you log in on the system. If, for any reason, the assignment does not exist or must be modified, :WORK: must be assigned to a write-enabled directory using the ASSIGN or the LNAME commands.

D.10 Related Publications

For more information on the Series IV Microcomputer Development System, see the following manuals:

- *Intellec® Series IV Microcomputer Development System Overview*, order number 121752
- *Intellec® Series IV Operating and Programming Guide*, order number 121753
- *Intellec® Series IV ISIS-IV User's Guide*, order number 121880

APPENDIX E USING AEDIT-86 ON THE iRMX™-BASED SYSTEM

AEDIT-86 runs on the iRMX operating system.

The iRMX operating system may be used with various terminals. Please check the characteristics of your particular environment and/or terminal.

E.1 Invocation Command

The invocation line is as described in Chapter 4.

If you are starting the tutorial in Chapter 1, type the following:

```
AEDIT <RETURN>
```

E.2 Special Considerations

Note that under iRMX-86 releases 5 and 6, you must specify AT=F.

E.3 Default Macro File

If AEDIT-86 is not located in the current directory, you may have problems activating the default macro file.

The name of the default macro file is AEDIT.MAC, which is assumed to be in the current directory. If it is not, you must explicitly specify the MACRO(*filename*) control.

E.4 Work File

The predefined file :WORK: must be properly assigned when AEDIT-86 is invoked. This assignment is usually done automatically when iRMX is booted. You must take care not to destroy :WORK:.

APPENDIX F CONFIGURING AEDIT-86 FOR OTHER TERMINALS

F.1 Tested Configurations

This appendix contains configurations for several other terminals. The terminals described here are not the only ones on which you can use AEDIT-86; they are merely the ones that have been tested. The following sections list the configuration functions and values required to run AEDIT-86 on the Intel-tested terminals. Configuration files for the following terminals are supplied with the AEDIT-86 program. The terminals are listed alphabetically according to the macro file name:

- 1510E.MAC—Hazeltine 1510 with escape lead-in
- 1510T.MAC—Hazeltine 1510 with tilde lead-in
- ADM3A.MAC—Lear Seigler ADM-3A
- MICROB.MAC—Beehive Mini-Bee
- QVT102.MAC—Qume QVT102, in VT102 mode
- S3.MAC—Series III with integrated Series III terminal
- S3E.MAC—Series III with integrated Series IIIE terminal
- S3ET.MAC—Series IIIE terminal used as terminal to remote host
- S4.MAC—Series IV terminal
- TV910P.MAC—Televideo 910 Plus
- TV925.MAC—Televideo 925 and 950
- VIEW3A.MAC—ADDS Viewpoint 3A Plus
- VT100.MAC—Digital Equipment Corporation VT100, VT101, VT102
- VT52.MAC—Digital Equipment Corporation VT52
- ZENTEC.MAC—Zentec Zepher and Cobra

Most CRT terminals have switches to set certain screen or keyboard characteristics. These switches must be set as in Table F-1 for AEDIT-86 to function correctly.

Table F-1. Switch Settings

Switch	Setting
BAUD RATE	Must match system. Use the maximum baud rate possible.
FULL DUPLEX	ON
RS232	ON
COMMUNICATION	CONVERSATIONAL
SELF ECHO	OFF
PARITY	INHIBIT if available, otherwise SPACE or 0 (ZERO)
PARITY SENSE	EVEN or ODD (don't care)
BITS/CHAR	8
STOP BITS	1
SCROLLING	ON
EOM (End Of Message)	OFF
AUTO LINE FEED	OFF
or EOL CHAR	CR ONLY
or NEW LINE	CR ONLY
or RETURN	CR ONLY
DTR	OFF
CHARS/LINES	80-255
LINES/PAGE	5-66
Xon/Xoff protocol should be disabled, if available.	
XON/XOFF	OFF
Wraparound must correspond to the AW configuration command.	
WRAPAROUND	ON if AW=T, or OFF if AW=F
Unique to the Lear Seigler ADM3A	
SPACE/ADVANCE	SPACE (destructive space)
< CTRL-Z > CLEAR SCREEN	ENABLE
AEDIT-86 don't care settings	
SCROLL TYPE	JUMP or SMOOTH
25TH LINE	
CHARACTER SET	
CURSOR STYLE	UNDERLINE, BLOCK, STEADY, BLINKING
AUTOREPEAT	
MARGIN BELL	
KEYCLICK	
SCREEN BACKGROUND	NORMAL, REVERSE
UPPERCASE ONLY or UPPER-/LOWERCASE	

1510E.MAC—Hazeltine™ 1510

This terminal displays 24 lines of 80 characters per line. The characters are generated in a 7×10 dot matrix. The maximum transmission rate is 19.2K baud. You may choose between the ESC or the tilde (~) character as the control sequence lead-in. It is advisable to use the tilde; if you use the ESC, you must change the BREAK character.

Macro file: 1510E.MAC – for the Hazeltine 1510 with escape lead-in.

```
AV=24;
AFCU= 0C; AFCD= 0B; AFCR=10; AFCL=8; AFCH= 12;
AFMU=1B 0C; AFMD=1B 0B; AFMR=10; AFML=8; AFMH=1B 12;
AFES=; AFER=1B 18; AFEK=; AFEL=1B 0F;
AFDL=1B 13; AFIL=1B 1A;
AFAC=1B 11; AO=0; AX=T;
AFRV=1B 1F; AFNV=1B 19; AC=T; AI=T;
ADDL=20; ADIL=20; \* 19200 baud *\

AB=7E; AFIG=1B;
AFXH=16;
```

Notes: Tilde is used for ‹ESC›.
‹CTRL-V› is used for ‹HEX›.

1510T.MAC—Hazeltine™ 1510

Macro file: 1510T.MAC for Hazeltine 1510 with tilde lead-in.

```
AV=24;
AFCU= 0C; AFCD= 0B; AFCR=10; AFCL=8; AFCH= 12;
AFMU=7E 0C; AFMD=7E 0B; AFMR=10; AFML=8; AFMH=7E 12;
AFES=; AFER=7E 18; AFEK=; AFEL=7E 0F;
AFDL=7E 13; AFIL=7E 1A;
AFAC=7E 11; AO=0; AX=T;
AFRV=7E 1F; AFNV=7E 19; AC=T; AI=T;
ADDL=20; ADIL=20; \* 19200 baud *\

AFIG=7E;
AFXH=16;
```

Notes: ‹CTRL-V› is used for ‹HEX›.

ADM3A.MAC—Lear Siegler™ ADM3A

This terminal displays 24 lines of 80 characters per line. The characters are generated in a 5×7 dot matrix. The maximum transmission rate is 19.2K baud.

Macro file: ADM3A.MAC for the Lear Siegler ADM3A.

```
AV=24;
AFCU=0B; AFCD=0A; AFCR=0C; AFCL=08; AFCH=1E;
AFMU=0B; AFMD=0A; AFMR=0C; AFML=08; AFMH=1E;
AFES=1A; AFER=; AFEK=; AFEL=;
AFAC=1B 3D; AO=20; AX=F;
ADES=5; \* 19200 baud *\

AFXH=16;
```

Notes: ‹CTRL-V› is used for ‹HEX›.

Switch	Setting
SPACE/ADVANCE	SPACE
‹CTRL-Z› CLEAR SCREEN	ENABLE

MICROB.MAC—Beehive™ Mini-Bee

This Beehive terminal can be formatted to display either 12 or 25 lines of 80 characters per line. Only the 25-character format is usable with AEDIT-86. Each character is generated in a 5×7 dot matrix. The maximum transmission rate for this terminal is 9600 baud. Note that the ESCAPE character has to be changed so that the default ESCAPE code can be used; <CTRL-K> is typed instead of escape.

Macro file: MICROB.MAC for the Beehive Mini-Bee.

```
AV=24;
AFCU=1B 41; AFCD=1B 42; AFCR=1B 43; AFCL=1B 44; AFCH=1B 48;
AFMU=1B 41; AFMD=1B 42; AFMR=1B 43; AFML=1B 44; AFMH=1B 48;
AFES=; AFER=1B 4A; AFEK=; AFEL=1B 4B;

AB=0B;
```

Notes: <CTRL-K> is used for <ESC>

QVT102.MAC—Qume™ QVT102

This terminal displays 24 lines with 80 characters per line. The maximum transmission rate is 19.2 K baud.

Macro file QVT102.MAC for the Qume QVT102 in VT102 mode.

```
AV=24;
AFCU=0B; AFCD=0A; AFCR=0C; AFCL=08; AFCH=1E;
AFMU=0B; AFMD=0A; AFMR=0C; AFML=08; AFMH=1E;
AFES=1B 2B; AFER=1B 59; AFEK=; AFEL=1B 54;
AFDL=1B 52; AFIL=1B 45;
AFAC=1B 3D; AO=20; AX=F;
ADIL=140; \* 19200 BAUD *\

AFXH=16;
```

Notes:: <CTRL-V> is used for <HEX>.

S3.MAC—Series III with Integrated Series III Terminal

Macro file: S3.MAC for the Series III with the integrated Series III terminal.

```
AH=S3;
```

S3E.MAC—Series III with Integrated Series IIIE Terminal

Macro file: S3E.MAC for the Series III with the integrated Series IIIE terminal.

```
AH=S3E;
```

S3ET.MAC—Series III as Terminal to Remote Host

Macro file: S3ET.MAC for the integrated Series IIIE terminal when used as terminal to a remote host.

```
AH=S3ET;
```

S4.MAC—Series IV Terminal

Macro file: S4.MAC for the Series IV terminal.

```
AH=S4;
```

TV910P.MAC—Televideo™ 910 Plus

This terminal displays 24 lines with 80 characters per line. The maximum transmission rate is 19.2K baud.

Macro file: TV910P.MAC for the Televideo 910 Plus.

```
AV=24;
AFCU=0B; AFCD=16; AFCR=0C; AFCL=08; AFCH=1E;
AFMU=0B; AFMD=16; AFMR=0C; AFML=08; AFMH=1E;
AFES=1B 2B; AFER=1B 59; AFEK=; AFEL=1B 54;
AFDL=1B 52; AFIL=1B 45;
AFAC=1B 3D; AO=20; AX=F;
ADIL=20; \* 19200 BAUD *\

AFXH=16;
```

Notes: ‹ CTRL-V › is used for ‹ HEX ›.

TV925.MAC—Televideo™ 925 and 950

This terminal displays 24 lines with 80 characters per line. The maximum transmission rate is 19.2K baud.

Macro file: TV925.MAC for the Televideo 925 and 950.

```
AV=24;
AFCU=0B; AFCD=0A; AFCR=0C; AFCL=08; AFCH=1E;
AFMU=0B; AFMD=0A; AFMR=0C; AFML=08; AFMH=1E;
AFES=1B 2B; AFER=1B 59; AFEK=; AFEL=1B 54;
AFDL=1B 52; AFIL=1B 45;
AFAC=1B 3D; AO=20; AX=F;
\* AFRV= 1B 47 34; AFNV= 1B 47 30; AC=F; AI=F; *\
ADIL=60; \* 19200 baud *\

AFXH=16;
```

Notes: ‹ CTRL-V › is used for ‹ HEX ›

VIEW3A.MAC—ADDS™ Viewpoint 3A Plus

This terminal displays 24 lines with 80 characters per line.

Macro file: VIEW3A.MAC for ADDS Viewpoint 3A Plus, in 3A plus mode.

```
AV=24;
AFCU=0B; AFCD=0A; AFCR=0C; AFCL=08; AFCH=1E;
AFMU=0B; AFMD=0A; AFMR=0C; AFML=08; AFMH=1E;
AFES=1B 2A; AFER=1B 59; AFEK=; AFEL=1B 54;
AFAC=1B 3D; AO=20; AX=F;
```

```
AFXH=16;
```

Notes: <CTRL-V> is used for <HEX>.

Switch	Setting
<CTRL-Z> CLEAR SCREEN	ENABLE

VT100.MAC—DEC VT100

This terminal can be formatted with 14 lines of 132 characters per line or 24 lines of 80 characters per line. Only the 24-line format is compatible with AEDIT-86. The characters are generated in a 7×9 dot matrix. The maximum transmission rate is 19.2K baud. You may choose between the DEC VT52 compatible and the ANSI standard (X3.41-1974, X3.64-1977) compatible terminal escape sequences for cursor control and screen erase functions. See the DEC VT52 description for the VT52 codes. Note that the ESCAPE character has to be changed so that the default ESCAPE code can be used; PF4 is typed instead of escape. This terminal does not have a HOME key; PF1 is typed instead of home.

Macro file: VT100.MAC for the Digital Equipment Corporation VT100, VT101, VT102.

Configuration commands for the VT100 are given in Table 9-1.

```
AH=VT100;
```

Notes: PF4 is set as escape.
PF1 is set as home.
Wraparound must be turned off.
Do not use baud rates above 9600.
Automatic XON/XOFF must be turned off.
The delete/insert line output function codes, although available on VT100 and VT102, are not used because of poor performance.

VT52.MAC—DEC™ VT52

This terminal displays 24 lines of 80 characters per line. The characters are generated in a 7×9 dot matrix. The maximum transmission rate is 19.2K baud. Note that the ESCAPE character has to be changed so that the default ESCAPE code can be used; <CTRL-K> is typed instead of escape. This terminal does not have a HOME key; <CTRL-O> is typed instead of home.

Macro file: VT52.MAC for the Digital Equipment Corporation VT52.

```
AV=24; AW=F;
AFCU=1B 41; AFCD=1B 42; AFCR=1B 43; AFCL=1B 44; AFCH=0F;
AFMU=1B 41; AFMD=1B 42; AFMR=1B 43; AFML=1B 44; AFMH=1B 48;
AFES=; AFER=1B 4A; AFEK=; AFEL=1B 4B;
AFDL=; AFIL=1B 49;
AFAC=1B 59; AO=20; AX=F;
```

```
AB=0B;
```

Notes: <CTRL-K> is used for <ESC>.
<CTRL-O> is used for <HOME>.

ZENTEC.MAC—Zentec™ Zepher and Cobra

This terminal displays 24 lines of 80 characters per line. The maximum transmission rate is 19.2K baud. To rub out a character on this terminal you must use SHIFT plus ESC.

Macro file: ZENTEC.MAC for the Zentec Zepher and Cobra.

```
AV=24;
AFCU=0B; AFCD=0A; AFCR=0C; AFCL=08; AFCH=1E;
AFMU=0B; AFMD=0A; AFMR=0C; AFML=08; AFMH=1E;
AFES=1B 2B; AFER=1B 59; AFEK=0D 1B 54; AFEL=1B 54;
AFDL=1B 52; AFIL=1B 45;
AFAC=1B 3D; AO=20; AX=F;
AFRV=1B 47 34; AFNV=1B 47 30; AC=F; AI=F;
ADIL=60; \* 19200 baud * \

AFXH=16;
```

Notes: The DEL key (shift ESC) is used for ‹RUBOUT›.
‹CTRL-V› is used for ‹HEX›.

APPENDIX G
ASCII CODES

ASCII Character	HEX	ASCII Character	HEX
NUL (↑@)	00	@	40
SOH (↑A)	01	A	41
STX (↑B)	02	B	42
ETX (↑C)	03	C	43
EOT (↑D)	04	D	44
ENQ (↑E)	05	E	45
ACK (↑F)	06	F	46
BEL (↑G)	07	G	47
BS (↑H)	08	H	48
HT (↑I)	09	I	49
LF (↑J)	0A	J	4A
VT (↑K)	0B	K	4B
FF (↑L)	0C	L	4C
CR (↑M)	0D	M	4D
SO (↑N)	0E	N	4E
SI (↑O)	0F	O	4F
DLE (↑P)	10	P	50
DCI (↑Q)	11	Q	51
DC2 (↑R)	12	R	52
DC3 (↑S)	13	S	53
DC4 (↑T)	14	T	54
NAK (↑U)	15	U	55
SYN (↑V)	16	V	56
ETB (↑W)	17	W	57
CAN (↑X)	18	X	58
EM (↑Y)	19	Y	59
SUB (↑Z)	1A	Z	5A
ESC	1B	[	5B
FS	1C	\	5C
GS	1D	]	5D
RS	1E	∧ or ↑	5E
US	1F	—	5F
space	20	`	60
!	21	a	61
"	22	b	62
#	23	c	63
$	24	d	64
%	25	e	65
&	26	f	66
'	27	g	67
(	28	h	68
)	29	i	69
*	2A	j	6A
+	2B	k	6B
,	2C	l	6C
−	2D	m	6D
.	2E	n	6E
/	2F	o	6F
0	30	p	70
1	31	q	71
2	32	r	72
3	33	s	73
4	34	t	74
5	35	u	75
6	36	v	76
7	37	w	77
8	38	x	78
9	39	y	79
:	3A	z	7A
;	3B	{	7B
<	3C	|	7C
=	3D	}	7D
>	3E	~	7E
?	3F	DEL	7F

NOTE: Control key equivalences are in parentheses.

INDEX

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Intel Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

____

____

____

____

____

____

2. Does the publication cover the information you expected or required? Please make suggestions for improvement.

____

____

____

____

____

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

____

____

____

____

____

____

4. Did you have any difficulty understanding descriptions or wording? Where?

____

____

____

____

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). ____________

NAME ______________________________ DATE ____________

TITLE ____

COMPANY NAME/DEPARTMENT ______________________________

ADDRESS ____

CITY ________________ STATE ____________ ZIP CODE ____________

(COUNTRY)

Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS ...

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.

intel®

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) **987-8080**

Printed in U.S.A.