underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  c = at sign;  u = question mark

CHAPTER 4

THE LINE AND SCREEN EDITOR

This chapter provides an overview of the editor and the two ways it is used.
Line-editing operation, screen editing operation, and editing command
descriptions are discussed in detail.

The editor is the ALDS program which provides for the entry and storage of
text.  The editor accepts keystroke entry and modification of lines of text
which compose, for example, assembly language programs or linker commands, and
it stores the lines in internal random-access memory (RAM).  The text in
memory can be written to a file on an external mass-storage device such as a
Wafertape™ device.  Conversely, a file from such a device can be read into
the CC-40 for editing.

The editor provides two methods of editing: line editing and screen editing.
Line editing uses the 31-character LCD display and keyboard in the CC-40
console, and it provides single-line display for line-number oriented text
entry and display.  Screen editing uses a terminal or a CRT display device
connected to the HEX-BUS™ interface through an RS232 device or a Video
Interface peripheral to provide cursor-oriented screen editing.  Screen
editing uses either the CC-40 keyboard or an external full-size keyboard for
text entry.

SYSTEM REQUIREMENTS
In a minimal CC-40 system configuration, line-editing operation requires that
at least one mass-storage device, such as a Wafertape™ device, is connected
to the CC-40.  The mass-storage device provides storage and retrieval for any
text which is stored in RAM.

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  ⌐ = at sign;  u = question mark

Line-editing operation with a minimal CC-40 configuration permits fully functional assembly-language program development through the use of the console display. The portion of the text shown by the display, however, is limited to a 31-character segment of a single line. Also, because the display shows only one line at a time, line numbers are necessary for keeping track of multiple lines in relation to each other.

More complete configurations, however, enhance editor performance by enabling the screen editing option. Significant improvement in editor performance, flexibility, and convenience results from including an RS232 device and a terminal or an external display for screen editing.

Screen editing with an external display provides longer display lines (typically 80 characters), and it provides multiple-line display (typically 24 lines). With screen editing, line numbers are largely unnecessary for referencing lines during text editing. Lines of text can be viewed in spatial sequence and addressed graphically on the display screen by a cursor. The logic of program statements is easily studied on the multiple lines simultaneously displayed. Additionally, if a CRT terminal (display combined with keyboard) is used, the full-sized keyboard permits more rapid, accurate, and convenient entry of keystrokes than does the compact keyboard of the CC-40.

## RUNNING THE EDITOR

After the ALDS is entered from BASIC with a [RUN "ALDS"] command, the ALDS program menu appears in the LCD display as shown below.

E)dit A)ssemble L)ink B)asicu

The editor is entered when [E] is pressed. When the editor begins to run, the display shows the editor menu as follows.

L)ine S)creen D)efine Q)uitu

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  c = at sign;  μ = question mark

The four functions which appear in the display make up only a part of the
menu.  Two other functions are scrolled into the display when either the
[SPACE BAR] or [μ] is pressed, as shown below.

    C)runch  U)ncrunchμ

## Line Editing:  "L" Option

Pressing [L] causes the editor to enter line-editing operation.  The editor
prompts with Copy file to determine if any file is to be copied from a
mass-storage device into RAM for editing.  Pressing [[ENTER]] in response to
the prompt indicates that no file is to be copied.  Typing in a device number
and a mass-storage filename (separated by a period, as in [1.FILE]) before
pressing [[ENTER]] loads the file of that name from the device.  After
[[ENTER]] is pressed (and a file is loaded if a name has been entered), the
editor places the command prompt ">" in the otherwise blanked display.

## Screen Editing:  "S" Option

Pressing [S] causes the editor to enter screen editing mode.  The screen of
the CRT is cleared, an asterisk and the end-of-file abbreviation (*EOF) appear
at the upper left of the screen to indicate that no text is in RAM, and a Copy
file prompt appears on the bottom line of the screen.  A file can be copied
into RAM from a mass-storage device such as a Wafertape unit by entering a
device number, a period, and a filename.  Pressing [[ENTER]] alone initializes
the editor for keyboard entry of lines of text.  Subsequently, the prompt
disappears, and the cursor appears in the upper left corner of the screen.

## Screen Definition:  "D" Option

Pressing [D] causes the editor to enter the screen-definition option.  In
screen definition, a mass-storage file can be used to install or customize the
editor for particular characteristics of the terminal or display to be used.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

Screen editing relies on a number of CRT-display characteristics, such as
communication parameters (data-transfer speed, etc.), screen size (number of
columns and lines), display-control commands (for cursor positioning, clearing
the screen, and so on). In addition, if screen editing is to use a terminal
keyboard instead of the CC-40 compact keyboard, particular terminal keys are
used to produce command codes (unprintable ASCII control characters or
characters following ASCII "escape") for issuing editing commands. Both
display and keyboard characteristics are defined during screen-definition
operations. Screen definition can (1) modify (or create) a table of display
and keyboard characteristics, (2) load a previously saved file of display and
keyboard characteristics for use during editing, or (3) save a table of
displayed keyboard characteristics to a file.

Two sets of default characteristics are stored in ALOS firmware. One set
contains screen and keyboard characteristics of the Televideo 920 terminal.
The other set contains screen definitions for a HEX-BUS™ Video Interface
device and key definitions for the CC-40 keyboard.

## Uncrunched Format Text: "U" Option
Pressing [U] causes the editor to operate in "uncrunched" mode. The display
then contains the message UNCRUNCHED MODE. Pressing any key causes the edit
menu to once again be displayed. With the editor in uncrunched mode, any text
typed into RAM or any file copied into RAM is represented in ASCII character
codes with null bytes to separate lines. No end-of-line characters such as
carriage returns or line feeds are used in the text. If a "crunched" (see
below) file is copied into RAM while the editor is in uncrunched mode, text
from the file is automatically converted to uncrunched text after it is read
into memory.

In uncrunched mode, no data-compression techniques are used to conserve memory
and decrease file-transmission time. Uncrunched mode is the default mode of
the editor; therefore, pressing [U] has no effect upon the editor unless
compressed or "crunched" mode (see below) has previously been selected.

underlining in text (except neads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  ⌐ = at sign;  ы = question mark

Selection of uncrunched text storage remains in effect until cruncned storage
is selected while the edit menu is displayed.  Text written to mass-storage
files wnile uncrunched mode is selected is written as it is stored in memory.

Cruncned Format Text: "C" Ootion

Pressing [C] causes the editor to store text in RAM in compressed or
"cruncned" format. The display then contains the message CRUNCHED MODE.
Pressing any key causes the edit menu to once again be displayed.  In crunched
mode multiple-character assembly-language mnemonic codes (keywords) are stored
as single byte "tokens," spaces surrounding keywords are suppressed, and
multiple spaces within a line are replaced with space suppression coces.
Other line characteristics are the same as in uncrunched format. The use of
crunched format results in a savings in file memory space and file
transmission time.

An uncrunched-mode file copied into memory while the editor is in crunched
mode is automatically converted to crunched text after it is read into memory.

Cruncned format remains in effect until either (1) the mode of storage is
changed wnile the edit menu is displayed or (2) operation of tne ALDS is
terminated and the CC—10 returns to BASIC.

Quitting the Editor:  "Q" Ootion

Pressing [Q] causes an exit from the editor menu.  The ALDS menu returns to
the display.

LINE-EDITOR OPERATION

After line editing is entered by pressing [L] in response to the edit menu and
the Copy file prompt receives a resoonse,  the command prompt ">" appears in
the display to indicate that the editor is ready to receive a command.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  ∆ = percent sign:  g = at sign:  u = question mark

## Text Lines and Line Numbers

The editor references each line of text with a line number. The command to
enter a line in response to the ">" prompt is a line number followed by a
space, the text of the line, and [[ENTER]]. The line number can be any five
digit number from 00001 through 32767; leading zeros are provided by the
editor if they are not entered at the keyboard. If a line with the number
typed is already present, the old line is replaced by the new line.

Entering a line number followed by no text inserts a blank line of text into
memory.

Editor line numbers provide the means of accessing particular lines. With the
command prompt ">" in the display, lines of text are called into the display
by line number for editing, insertion, and deletion.

Line numbers are not stored with the text in memory. They are computed by the
editor each time a line is displayed. The effect of entering [1000 This] as
the initial line in the text area of memory is to store the text [This] in
memory for display as 00001 This in any subsequent display. Typing in another
line with the number 1000 produces a text line numbered 00002. A displayed
line number (always with a trailing space) indicates the position of a line
relative to the beginning of text. The number of any line changes as the
number of lines between it and the first line of the text changes.

## Line Editing Commands

Commands available for use in line editing are listed below. Where a
parenthesis sets off the last characters of a command, only the first
characters need be typed to issue the command. Detailed descriptions of
commands appear on the pages noted in the list.

AUTO--Sets up the editor to produce a line number for the first line and
to produce a new line number after each line is entered.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed: double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  ₃ = at sign:  u = question mark

COPY—Loads a file from mass-storage (Wafertape device, etc.) into memory
for subsequent line editing.

DEL(ETE)—Deletes a line or range of lines from the text in memory.

E(DIT)—Displays a line of text for line editing.

F(IND)—Finds the next character string that matches the string specified
in the command.

FORMAT—Initializes the mass-storage medium in the specified device.

LINE—Sets the editor so that it displays line numbers with each line in
the display.

LIST—Shows lines of text in memory in the display or transfers the lines
to an external unit such as a printer.

MOVE—Moves a line or a range of lines to another location in the text.

NOLINE—Sets the editor so that it does not display line numbers with each
line in the display.

QUIT—Terminates line editing and returns to the main edit menu.

R(EPLACE)—Replaces one or more occurrences of a character string in the
text with another specified string.

SAVE—Saves the text currently in memory to a mass-storage device with the
specified device number and filename.

underlining in text (except heads)- indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

UNDO—Cancels the effects of the last delete command, line modification,
or replacement operation by reinserting into the text the lines-deleted or
changed.

VERIFY—Compares character-for-character the text stored in memory with a
file present in an external storage device.

## New Text Entry

When the AUTO command is typed in response to the ">" prompt, the editor is
set to number each line automatically.  The LCD display shows the number 00001
followed by a space and the blinking cursor.  For normal keystroke entry of a
line, the line is typed and the cursor advances to the next position to the
right after every keystroke.  When [[ENTER]] is pressed, the line is moved
from the display buffer into the text-storage area of memory.

The display buffer is then filled with spaces, and the line number 00002 and a
space appear in the display.  Successive lines are always numbered in
increments of 1.  In automatic line-numbering operation, for example, the line
editor always displays a line following the entry of line 00002 with the
number 00003, a line following line 00053 with the number 00054, and so forth.

The [[BREAK]] key terminates AUTO and returns the ">" command prompt to the
display.

## Editing Previously Entered Lines

When lines are already present in the text area of memory and the ">" prompt
is displayed, the editor displays any line upon entry of a command of the
format [E] [NNNNN] [[ENTER]].  For example, the command line E 12 [[ENTER]]
causes the editor to read the 12th line of the text into the display buffer.
When [[ENTER]] is pressed after the line is edited, the edited line is stored
in the text area of memory and the ">" command prompt returns to the display.

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed

alpha = pound sign; Δ = percent sign; c = at sign; u = question mark

---

If, while a line is being displayed, the [Up Arrow] key is pressed, the line currently in the display buffer is transferred into the text area of memory. Then the line with the next lower number is displayed. In effect, the text is scrolled backward through the display window by one line.

Similarly, if the [Down Arrow] key is pressed, the display-buffer line is transferred to the text area, and the text line with the next higher number is displayed. In effect, the text is scrolled forward through the display window.

## Using the Display Buffer

Whenever a numbered line is present in the display buffer after entering an EDIT or LIST command, the buffer functions as a scratchpad for entry of the line into the text area of memory. When [[ENTER]], [Up Arrow], or [Down Arrow] is pressed, the display-buffer line is transferred to memory and the adjacent line is transferred from memory into the display buffer. Whenever [[BREAK]] is pressed while a numbered line is in the display buffer, the buffer is cleared and the ">" command prompt returns, but the line from the display is not stored in memory.

If a line number (followed by a space) in the display buffer is altered, then the new number determines the point of entry into the text area of memory after the line has been entered.

If the space following the line number is replaced by a [-] (minus sign), then the line is inserted into the text area as a new line before the one currently with that number. If a [+] (plus sign) replaces the space, the line is inserted into the text as a new line immediately after the line with the number displayed.

## Clearing the Display Line

The contents of the display buffer are cleared when [[CLR]] is pressed. A new line can then be typed into the display buffer.

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
aloha = pound sign;   & = percent sign;   @ = at sign:   ? = question mark

The prompt ">" does not appear in the cleared display, although the cleared line is used in exactly the same manner as the command line on which the prompt appears.  In command mode (where a line originating with the prompt has been cleared), the lack of a prompt is the only result of pressing [[CLR]] rather than [[BREAK]].  The display returns the command prompt ">" when a new line is entered.

In text entry during AUTO mode, however, the difference between [[CLR]] and [[BREAK]] has the added significance that [[CLR]] returns the editor to AUTO mode after the new line is entered while [[BREAK]] causes the editor to reenter command mode.  A new numbered line of text entered during AUTO mode causes the line-numbering sequence to be reset to the number of the new line, and the editor continues in AUTO mode.  However, if a command is entered in the cleared display and executed during AUTO mode, the editor exits AUTO and reenters command mode upon completion of the command.

## Using the Playback and Recover Buffers

Two buffers in the CC-40 save text for repeated entry of commands or cancellation of delete commands, text modification, or replacement performed in error.

The playback buffer always contains the last command line entered into the editor.  Pressing [[SHIFT]] [Up Arrow] causes the contents of the playback buffer to be transferred to the the display buffer.  Through the use of the playback buffer, an editor command such as FIND can be executed numerous times without being retyped.  The command is entered into the display buffer only once through the keyboard, and thereafter the command is transferred into the display buffer from the playback buffer.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;   a = percent sign;   y = at sign;   u = question mark

Similarly, the recover buffer always contains the line or lines most recently
deleted from the text area of memory, the original text of the line most
recently modified, or the line last replaced.  The editor, by referencing the
lines in the recover buffer, can cancel the most recent deletion or
line-editing operation.  When UNDO is entered, the lines in the recover buffer
are reinserted into the text area in their original position.

## Line-Number Display Commands

The NOLINE command eliminates the display of line numbers, providing a greater
number of text characters to be viewed at the beginning of each line.  The
NOLINE command has the effect of shifting the characters of the display line
six character-positions to the left (corresponding to a five-digit line number
followed by a space).  The line number, even though not in view, is usable for
line-indexing operations.  For example, setting the editor for NOLINE
operation and then typing in a command to edit line 15 still brings line 15
into the display; the only difference is that the line number cannot be seen
without forcing the cursor against the left side of the display window into
the line.

The LINE command causes the line numbers to once again automatically appear in
the display.

## Cursor Control

When a line is in the display, five keys control cursor position in the
display.  The cursor can be moved to the right or to the left one character,
it can be moved to preset tab stops, and it can be moved to the first position
of the line (home).

The [Right Arrow] key moves the cursor one character to the right of its
current position.  The [Left Arrow] key positions the cursor one character to
the left. When either key is held down, repeated one-character cursor movement
begins; when the key is released, cursor motion stops.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  ɔ = at sign;  u = question mark

The left- and right-arrow keys have no effect if the cursor is at,
respectively, the the beginning or end of the 30-character line.  If the
cursor moves to a character position not in the display, the display window is
shifted one character in the direction of the motion of the cursor.
The forward tab key [[CTL]] [Right Arrow] moves the cursor to the closest tab
stop to its right.  If the cursor is to the left of character position 25,
pressing the key moves it to position 25.  If the cursor is between character
positions 25 and 50, pressing the key moves it to position 50.  Pressing the
forward tab key when the cursor is beyond position 49 has no effect.

The back-tab key, [[CTL]] [Left Arrow], moves the cursor to the closest tab
stop to its left.  If the cursor is to the right of character position 50,
pressing the key moves it to position 50.  If the cursor is between character
positions 25 and 50, pressing the key moves it to position 25.  Pressing the
back-tab key when the cursor is to the left of character position 26 moves it
to position 1.

The home key, [[CTL]] [Up Arrow], regardless of the current cursor position,
moves the cursor to the first character of the line (the most significant
digit of the line number if a text line is being displayed).

Deletion of Text in the Display
Text in the display can be deleted character-by-character, erased from the
current cursor position to the end of the line, or completely cleared (as
described above).

The delete key, [[SHIFT]] [Left Arrow], deletes the single character at the
current cursor position and shifts all subsequent characters one position to
the left.

The erase-to-end-of-line key, [[CTL]] [Down Arrow], deletes all characters
from the one at the current cursor position to the end of the display line.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  c = at sign;  v = question mark

None of the delete display-control keys has any immediate effect on a line
already stored in the text area of memory. For example, if there is a line 30
which has been moved from memory into the display buffer and [[CLR]] is
pressed, line 30 remains in the text area of memory exactly as it was before
its being copied into the display buffer.  Line 30 in the text area of memory
is replaced by a line in the display buffer numbered 30 only after [Down
Arrow], [Up Arrow], or [[ENTER]] is pressed.

## Insertion of Characters into Displayed Text

Insertion of characters into the display buffer occurs after the insert
[[SHIFT]] [Right Arrow] key is pressed.  The editor is set to insert
characters at the current cursor position, moving all characters from the one
at the cursor position to the end of the line to the right.  Insertion of
characters continues until a subsequent line-editing control key
(cursor-position key or delete key) or line entry key ([[ENTER]], [Up Arrow],
or [Down Arrow]) is pressed.

If the number of characters in the display buffer (including the line number
and trailing space) exceeds 80 during insertion of characters, the line is
truncated with the 80th character.  In effect, characters in excess of the
80th "fall off the end of the line."

## Insertion of Single Lines of Text

From the command prompt, a line can be entered before or after a line
presently in memory by typing, respectively, a "-" (minus sign) or a "+" (plus
sign) in lieu of the space between the line number and the text of the line.
Entering [10-*] creates a new line 00010 which contains an asterisk, and the
old line 00010 becomes line 00011.  Entering [10-LABEL1] creates a new line 11
containing the word "LABEL1", line 00010 remains as it was before the entry,
and the previous line 00011 is renumbered to become line 00012.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

Using "+" and "-" in the column following the line number while a line is
being displayed in AUTO mode, is being edited after an EDIT command is given,
or is being listed after a LIST command is given has the same effect that it
does from command mode.  Automatic entry, editing, and listing, however,
continue until AUTO mode is terminated with [[BREAK]], editing is terminated
with [[ENTER]], and listing is terminated with [[BREAK]] or the end of the
text.

## Insertion of Multiple Lines of Text

With lines of text in memory, AUTO mode effectively produces line insertion.
Entering AUTO mode with the number of a line already present in memory causes
the number of the line followed by a trailing "+" (plus sign) to automatically
appear in the display.  When the text for the line is entered, the line is
assigned the next higher number, and that number followed by "+" appears in
the display.

With four lines of text already in memory, for example, entering [AUTO 3]
causes 00003+ to appear in the display.  Text entered becomes part of a new
line numbered "00004," and after entry 00004+ appears in the display for
additional insertion of lines.  The original fourth line remains the next line
of the text; its line number is incremented with each new inserted line.

If AUTO mode is entered at the last line of text, the display lines are
slightly different.  Again with four lines of text in memory, if AUTO mode is
entered with AUTO 4, the display contains 00004+.  When the text for the new
line is entered, the line is stored as line 00005.  The display then contains
00006 followed by a space, indicating that new lines entered are being
appended rather than inserted.

## Text Line Replication

A line of text currently in the display can be replicated into a line above or
below it by backspacing the cursor to the space following the line number and
pressing [+] or [-].

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

Entering a line in which [+] replaces the space trailing the line number
causes two changes to the text. First, the text of the previously displayed
line appears in the display with the next higher line number:  it has become
the second line with that text in memory. Second, numbers of lines following
the replicated line are incremented. Using this process, a line can be
replicated "after itself" any number of times by moving the cursor to the
space trailing the line number, pressing [+], and entering the line.

The [-] key performs the same function, except that the replicated line
becomes the line previous to the one which has been displayed. The replicate
has the same number as the original line, and the original line appears in the
text as the line with the next higher number. A line can be replicated before
itself any number of times by repeatedly moving the cursor into the space
trailing the line number, pressing [-], and entering the line.


## SCREEN-EDITOR OPERATION

In screen editing, an external terminal or display becomes a multi-line
"window" for viewing the text stored in CC-40 memory. All lines (rows) on the
screen except the last one are available for display of text lines. The last
line is reserved for display of editor commands entered through the keyboard.

Screen addressing (for both lines and columns) is performed by a
keyboard-controlled cursor. The cursor marks either the position at which
characters are entered by keystroke or the position at which the next
cursor-relative command or display-control function (delete character, for
example) takes effect. The cursor can be moved to any line or column of text
on the screen. For text in which the number of lines exceeds the number of
rows in the display, the lines of text displayed on the screen are selected by
either line number or page scrolling (moving the display window to the
previous or subsequent "screen" of lines).

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  α = at sign;  u = question mark

Any RS232-compatible terminal or external display terminal can be connected to
the CC-40 console through an RS232 device for screen editing.  Terminal or
display characteristics, however, must be defined for the editor.  These
characteristics include tne column and line format of the display screen, the
transmission characteristics used for RS232 communication, and the
display-control characters used to determine operations such as cursor
positioning.  The characteristics are stored into memory for use by the screen
editor with the D)efine option in the main edit menu.

The keyboard used for screen editing can be either the CC-40 compact keyboard
or another keyboard which is part of the terminal providing the display.  The
D)efine option also specifies which keyboard is used and defines the keys
which are to be used to issue editor commands.  Saving text as a mass-storage
file, for example, can be chosen to be the sequence [[CTL]] [S], [[ESC]] [S],
or some other key or key sequence, depending on user preference.

The D)efine option must be used before the S)creen option is used each time
the ALDS is entered from BASIC.  Screen and keyboard default values for the
Televideo 920 terminal and the internal keyboard and a HEX-BUS™ video
Interface display device are included in firmware, and through D)efine they
can be selected for use as they are or they can be modified for other
devices.  Alternately, previously saved sets of characteristics can be loaded
from a mass-storage device.  If neither of these steps is taken before the
S)creen option is selected from the main edit menu, the screen editor cannot
be used and the No screen parameter table message appears in the console
display.

## Screen-Editing Commands

Like line editing, screen editing uses display-function keys.  These keys are
pressed during text entry to cause cursor movement in columns and lines,
vertical paging (scrolling page-by-page), insertion and deletion of text
within lines, and other similar operations.  Also as in line editing, screen
editing uses commands.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;   Δ = percent sign;   τ = at sign;   u = question mark

In screen editing, however, the text remains on the screen while the commands
are typed.  In general, a two-keystroke sequence initiates command entry.  The
editor then prints out the full word or phrase for the command in the last
line of the display and places the cursor after the word or phrase.
Additional command information (containing line numbers, device numbers, and
filenames, for example) is then typed into the command line.

The command line can be edited using keys previously defined for clear-line,
delete-character, insert-character, left-and right-arrow, forward- and
back-tab, and erase-field keys.  All printable characters can be entered.
Pressing [[ENTER]] begins execution of the command.  Pressing [[COMMAND EXIT]]
clears the command line and returns the cursor to the text display area of the
screen.

Most of the commands and functions available in line editing are available in
screen editing.  Some additional commands are also available to make full use
of the greater speed and convenience of the screen display.

Entry of screen editing commands is, in the default command set for the
Televideo 920 terminal, initiated by escape sequences.  The nonprintable ASCII
character "ESC" is entered through the keyboard, followed by a character which
is generally the first letter of the full word or phrase which names the
command. Command keys in the CC-40 keyboard default set are similarly defined
for use following the [[FN]] key.  For example, entry of the delete-line
command requires the sequence [[ESC]] [D] on the Televideo 920 and [[FN]] [D]
on the CC-40 keyboard.

Screen editing commands are listed below.  Detailed descriptions of commands
are referenced on pages noted with each command.

    COPY--Copies a file from mass-storage (a Wafertape™ device, for
    example) into memory for subsequent editing.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign:  ⫫ = at sign:  ⎁ = question mark

DELETE--Deletes a line or range of lines from the text in memory.

FIND--Finds the next string of characters that matches the string
specified in the command.

FORMAT--Initializes the mass-storage medium in the device specified.

HELP--Displays a list of screen-editor commands and cursor-control
keystrokes on the screen.

JUMP TO LINE--Places the cursor on the line with the number specified in
the command.

LIST--Outputs the text in memory to a HEX-BUS™ device such as a printer.

MOVE--Moves a line or a block of lines to another location in the text.

QUIT--Leaves the screen editor and returns to the main editor menu.

REPLACE--Replaces one or more occurrences of a character string in the
text with another specified string.

SAVE--Saves the text currently in memory to a specified mass-storage
device.

TAB DEFINE--Redefines the display tab stops to be other than those set by
default.

UNDO--Cancels the effects of the last delete command, line modification,
or replacement operation by reinserting into the text the lines deleted or
changed.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign:   Δ = percent sign:   σ = at sign;   μ = question mark
_____

    VERIFY—Compares character-for-character the text stored in memory with a
    file present in an external storage device.


Text Entry

Text is entered by typing a line and pressing [[ENTER]].  After the line is
entered, the cursor moves to the first column of the text area of the
following line.


A special AUTO provision in the screen editor inserts a new blank line of text
each time [[ENTER]] is pressed.  Lines from the new blank one to the end of
text have their line numbers incremented by one.  AUTO mode permits new lines
to be inserted into text as easily as if they were being appended to the end
of text.


When the editor is first entered, automatic line-entry mode is in effect.
Pressing [[CTL]] [A] the first time turns it off, and pressing the key a
second time turns it on again.  Throughout editing, pressing the key toggles
insertion mode on whenever it is off and toggles it off whenever it is on.


Text is displayed after five-digit line numbers and a space, as in line
editing.  The line number and space in screen editing, however, are
inaccessible to the cursor.


Cursor Positioning

The cursor is stepped through the columns of the current (cursor) line with
the [Right Arrow] and [Left Arrow] keys.  The right-arrow key [Right Arrow]
positions the cursor one character to the right of its current position, and
the left-arrow key [Left Arrow] positions the cursor one character to the left.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  ☢ = at sign;  ч = question mark

If [Right Arrow] moves the cursor past the last character typed in the line
without a displayable character being typed, the character positions between
the last character and the cursor are ignored when the line is stored into the
text area of memory.  No trailing spaces are entered into the text.  If,
however, a printable character is typed after the skipped character positions,
then the positions are entered as spaces into the text.

The [Up Arrow] and [Down Arrow] keys perform similar functions in moving the
cursor one line up and one line down on the screen.

The home key ([HOME] on the terminal keyboard, [[CTL]] [Up Arrow] on the CC-40
keyboard) positions the cursor to the first character of the first line in the
display.

The forward tab key ([TAB] on the terminal keyboard, [[CTL]] [Right Arrow] on
the CC-40 keyboard) moves the cursor to the closest tab stop to its right.
The tab stops are those set either by the screen-editing definition file or by
the TAB DEFINE command during screen editing.  Pressing the forward tab key
when the cursor is to the right of the last tab stop moves the cursor to the
first column of the same display line.

The back-tab key ([[CTL]] [U] on the terminal keyboard, [[CTL]] [Left Arrow]
on the CC-40 keyboard) moves the cursor to the closest tab stop to its left.
Pressing the back-tab key when the cursor is to the left of the first tab stop
in a line moves the cursor to the last column of the same display line.

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed

alpha = pound sign;  ∆ = percent sign;  c = at sign;  u = question mark

## Display Paging Control

The page-forward key ([[ESC]] [Down Arrow] for the terminal and [[CTL]] [+] for the CC-40) and the page-back key ([[ESC]] [Up Arrow] for the terminal and [[CTL]] [-] for the CC-40) scroll the text lines up and down one screen page at a time.  When the page-forward key is pressed, the text line following the one in the last line of the text area becomes the first line of the new page. The display is effectively moved up "a page" (i.e., the number of lines within the text-display area of the screen).  Similarly, the page-back key scrolls the lines of the text one page downward.

## Screen Refresh (View)

The [[CTL]] [V] (view) key rewrites text lines to the current display.  If a terminal or display is cleared locally through keystroke or power interruption, view can rewrite the text display to the screen.

## Line-Number On/Off Toggle

The line-number display key [[CTL]] [T] controls the position of text lines on the display screen.  If no line numbers are to be displayed, the line is effectively shifted to the leftmost six columns—five line-number digits and a space—in either display so that only the typed-in portion of the line is displayed.  (Whether displayed or not, the six line-number columns count toward the maximum 80 characters permitted in each line.)

Initially, the editor displays line numbers.  During editing, pressing [[CTL]] [T] the first time causes the line-number columns to be shifted from view. Pressing the key a second time causes them to again be shifted into view.  The key operates as a toggle which causes the line numbers to alternately disappear and appear.

## Text Deletion

Displayed characters can be deleted by (1) single character, (2) all characters from the cursor position to the end of the line, and (3) all characters in the line.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;   ∆ = percent sign;  σ = at sign;   u = question mark

The delete key ([[DEL]] for the terminal, [[SHIFT]] [Left Arrow] for the
CC-40) deletes the single character at the current cursor position and shifts
all subsequent characters one position to the left.

The erase-to-end-of-field key ([[CTL]] [E] for the terminal, [[CTL]] [Down
Arrow] for the CC-40) deletes all characters from the one at the current
cursor position to the end of the display line.

The clear key ([[CTL]] [C] for the terminal, [[CLR]] for the CC-40) erases all
characters, except the line number and space trailing it, from the line.
Screen editing enters lines into the text area of memory differently from line
editing.  Lines displayed on the screen are not operated on as if they were
line-by-line scratchpads.  Pressing [[CLR]] during line editing for example,
clears the line number and results in only the display being cleared; the
current text line is not affected.  When the clear key ([[CTL]] [C] for the
terminal or [[CLR]] for the CC-40 keyboard) is pressed during screen editing,
a blank line remains on the screen (with the line number unless it has been
toggled off).

The screen editor, like the line editor, has a provision for recovery from
erroneous entries through the use of the recover buffer and the UNDO command
([[ESC]] [U] for the terminal, [[FN]] [U] for the CC-40).  If the text of a
line is accidentally cleared, for example, regardless of whether the cursor
remains on the line or is moved to some other text line, the cleared text is
restored to the line with UNDO.  In screen editing, as in line editing, the
recover buffer stores the original text of the last line modified or replaced,
or it stores the last line or block of lines deleted.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  @ = at sign;  u = question mark

## Text Insertion

Characters typed into the text area of memory after the character-insert key
([[CTL]] [Q] for the terminal, [[SHIFT]] [Right Arrow] for the CC-40 keyboard)
displace all other characters from the cursor to the end of line.  When the
character-insert key is initially pressed, the editor is set to insert any
characters at the current cursor position, moving all following characters to
the right.  If the text in the line reaches 74 characters, no further
characters are accepted into the line (they do not "fall off the end of the
line" as in line editing).  Character-insertion continues until a subsequent
display-control key (cursor-position or delete key or [[ENTER]]) is pressed.

## Screen-Editing Keyboard and Display Definition

Particular keyboard and terminal characteristics are defined by a table of
values stored in memory for use during screen editing.  The screen-definition
menu provides for selection of particular values used during editing with
three options.

Pressing [D] in response to the main edit menu causes the editor to display
the screen-definition menu shown below.

### M)odify S)ave L)oad O)uit

Pressing [M] in response to the screen editor menu causes the editor to begin
displaying a series of prompts for modification of current table values by use
of line editing and the console keyboard and display.  Normally, the values
displayed are the "default" values available in editor firmware: they are
values which work with a Televideo 920 terminal or with the console keyboard
and the HEX-BUS™ Video Interface device.  With each prompt, previous or
default selections can be retained by pressing [[ENTER]], or they can be
modified through the use of line editing functions.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  a = percent sign;  σ = at sign;  u = question mark

Pressing [Up Arrow] or [Down Arrow], respectively, causes the display to
"scroll" to the previous or following prompt.  Pressing [[BREAK]] in response
to any prompt returns the definition menu to the display (with characteristics
already modified retained in the screen parameter table in memory).

All modifications made to the table during screen definition are in effect in
the screen editor as long as the ALDS is running.  The editor can be tested by
exiting the screen-definition menu with [Q] in order to return to the main
edit menu.  Selecting S)creen from the main edit menu by pressing [S] starts
screen-edit operation with all new characteristics from the table in effect.

After new screen-editor characteristics have been tested, they can be saved to
a mass-storage file.  They can be conveniently reentered from the file instead
of the keyboard at a later time. Pressing [S], and entering the mass-storage
device number and filename saves the file.

Pressing [L] and entering a mass-storage device number and filename results in
the loading of any previously stored definition file for use or further
modification.

Note carefully that until the Define option is selected and either a file has
been loaded or the M)odify option has been selected to specify at least the
keyboard and screen default values, no table is present in RAM.  Attempting to
enter the screen editor from the main edit menu results in a No screen
parameter table message.

The table of values for screen editing requires the following information.

     !o! Selection of internal (CC-40) or terminal keyboard use

     !o! The device number and data-transmission characteristics (such as RS32
         parameters) of the HEX-BUS™ device connected to an external
         terminal or display

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;   Δ = percent sign;   σ = at sign:   u = question mark

:o! The specification of which keys on the keyboard (usually control

   characters or escape sequences) are used for screen-editor functions

   and commands


:o! The specification of terminal display characteristics (such as screen

   size and display-control codes)


Keyboard Selection
The editor first displays the prompt Internal keyboard (Y/N)u.  Pressing [Y]
selects the CC-40 keyboard and the associated default values for table entry.
Pressing [N] selects the external terminal keyboard with associated default
values.  The default values are loaded only if no screen parameter table
resides in memory.


Selection of Device and Data-Transmission Characteristics
The editor displays the prompt Screen Device for entry of display and any
terminal keyboard device data-transmission characteristics, such as the RS232
device code and transmission characteristics.  Device number, baud rate, no
carriage return with lines (R=N), wait mode (T=W), no echo (E=N) and no data
overrun reporting (O=N) must be provided.  Entry of other characteristics
depends on terminal requirements.  Shown below is an example entry setting up
RS232 characterics through device 20 to communicate with a terminal or
external display device set for a baud rate of 4300, seven-bit data transfer,
one stop bit, and even parity.  Pressing [[ENTER]] with these default values
displayed stores them into the RAM table from editor firmware.  At this prompt
or any subsequent one, pressing [[BREAK]] stores values previously modified
into the table along with the remaining default values.  The screen-definition
menu returns to the display.  For information on how to use the screen editor
with the HEX-BUS™ Video Interface device, see appendix D of the Reference
Manual.


    20.2=4300,R=N.T=W,O=N,P=E,E=N

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  υ = question mark

## Command Keystroke Definition

A series of prompts following the entry of the terminal characteristics provides for definition of command and function keys.  These definitions differ according to terminal design and user preference.  The codes produced by special terminal keys are usually listed in the user documentation for each terminal.

The prompts for the command keys contain the names of edit commands and functions (copy, delete, and the like) and the default values assigned to them.  Entry in response to each of the prompts is the value of the byte or bytes originating from the keyboard when the key is pressed.  The values of bytes can be expressed in either decimal or hexadecimal notation (for example, [[ESC]] = 27 or >1B).  Alphabetic characters can be expressed directly (without referencing their ASCII values) for entry in two cases: (1) when they are surrounded by double quotes, as in "Q", and (2) when they are preceded by the letters CTL to convert them to equivalent control-code values.  Typing in "Q" stores the ASCII value of the letter (>51 or 81), and typing in CTL A stores the value of CTL-A (>01 or 1).

Either one or two values can be typed in for each command or function definition.  If one value is typed, zero is assigned to the second value in the table.  If two values, separated by a comma, are typed, both values are entered into the table.

The specific commands for which keystrokes must be defined are as the following.

| Copy | Help | Quit | Undo |
| Delete | Jump | Replace | Verify |
| Find | List | Save | |
| Format | Move | Tab define | |

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign:  u = question mark
_____

Pressing [[ENTER]] to store values for the screen device number and

transmission characteristics, for example, brings the prompt shown below into

the console display.  The pairs of values shown are the default values for the

two keys which initiate the COPY command from either keyboard selected: they

are the decimal representation of the ASCII characters "ESC" and "C".


   Copy key: 27, 67
   _____


If these values are to be modified, the first keystroke for the new value(s)

entered clears the display and show the character entered.  Changing the

values to the single value of CTL-C (>03 or 3) requires that one of the

entries shown below appear in the display before [[ENTER]] is pressed.


   CTL C
   _____

   3
   _

   >3
   __


Function-Key Definition

Following the prompts for and entry of screen-editor commands are prompts for

defining display-function keys.  Display-function keys, like keys which can be

selected for commands, are dependent on terminal design, and the user

documentation for each terminal lists the codes which they produce.


Function-key definitions are modified from the default values in exactly the

same way command-key definitions are.  The specific display-functions which

must be defined are listed below.


| Auto          | Delete line  | Insert char   | Page forward  |
| Back tab      | Down cursor  | Insert line   | Right cursor  |
| Clear line    | Enter        | Left cursor   | Tab           |
| Command exit  | Erase field  | Line display  | Up cursor     |
| Delete char   | Home         | Page back     | View          |

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  % = percent sign;  @ = at sign;  ? = question mark

Default Values for Command and Display-Function Keys

Default values for commands and functions are represented in the following
table by the keys which produce them on both Televideo 920 and CC-40 keyboards.

### EDITOR DEFAULT KEYBOARD DEFINITION ENTRIES

| FUNCTIONS | KEYS USED | |
| OF | EXTERNAL KEYBOARD | INTERNAL KEYBOARD |
| KEYS | (Televideo 920) | (CC-40) |
| --- | --- | --- |
| Copy | [[ESC]] [C] | [[FN]] [C] |
| Delete | [[ESC]] [D] | [[FN]] [D] |
| Find | [[ESC]] [F] | [[FN]] [F] |
| Format | [[ESC]] [N] | [[FN]] [N] |
| Help | [[ESC]] [H] | [[FN]] [H] |
| Jump | [[ESC]] [J] | [[FN]] [J] |
| List | [[ESC]] [L] | [[FN]] [L] |
| Move | [[ESC]] [M] | [[FN]] [M] |
| Quit | [[ESC]] [Q] | [[FN]] [Q] |
| Replace | [[ESC]] [R] | [[FN]] [R] |
| Save | [[ESC]] [S] | [[FN]] [S] |
| Tab define | [[ESC]] [T] | [[FN]] [T] |
| Undo | [[ESC]] [U] | [[FN]] [U] |
| Verify | [[ESC]] [V] | [[FN]] [V] |
| Auto | [[CTL]] [A] | [[CTL]] [A] |
| Back tab | [[CTL]] [U] | [[CTL]] [Left Arrow] |
| Clear | [[CTL]] [C] | [[CLR]] |
| Command exit | [[CTL]] [X] | [[CTL]] [X] |
| Delete char | [[DEL]] (>7F) | [[SHIFT]] [Left Arrow] |
| Delete line | [[CTL]] [O] | [[CTL]] [O] |
| Down | [↓] ([[CTL]] [J]) | [↓] |
| Enter | [[ENTER]] ([[CTL]] [M]) | [[ENTER]] |
| Erase field | [[CTL]] [E] | [[CTL]] [↑] |

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  △ = percent sign;  c = at sign;  u = question mark

| | | |
|---|---|---|
| Home | [HOME] (>1E) | [[CTL]] [Up Arrow] |
| Insert char | [[CTL]] [Q] | [[SHIFT]] [Right Arrow] |
| Insert line | [[CTL]] [N] | [[CTL]] [N] |
| Left | [Left Arrow] ([[CTL]] [H]) | [Left Arrow] |
| Line display | [[CTL]] [T] | [[CTL]] [T] |
| Page back | [[ESC]] [Up Arrow] | [[CTL]] [-] |
| Page forward | [[ESC]] [:] | [[CTL]] [+] |
| Right | [Right Arrow] ([[CTL]] [L]) | [Right Arrow] |
| Tab | [TAB] ([[CTL]] [I]) | [[CTL]] [Right Arrow] |
| Up | [Up Arrow] ([[CTL]] [K]) | [Up Arrow] |
| View | [[CTL]] [V] | [[CTL]] [V] |

Definition of Required Display-Control Information

Following function-key definitions, the editor provides prompts for entry of
terminal or display screen size and display-control codes. Like control and
function key definitions, screen size and display-control codes are particular
to each terminal.  User documentation on each terminal lists the parameters to
be entered.  Values for some terminals are listed in appendix 8 of the CC-40
Editor/Assembler Reference Manual.

The prompts for screen size and display control codes, and the information
which is entered in response to them, are listed below.

Screen size--The number of rows (lines) available on the display and the
number of columns (character positions) in each line must be entered in
the following format: columns,rows (without a space following the comma).

Clear screen--The character code or codes used to clear the screen of
the terminal must be entered.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

Move cursor—The first character code or codes used to initialize the
sequence which positions (addresses) the cursor on the terminal must be
entered.  Following this byte or bytes in the sequence are two bytes
which specify the cursor addresses (row and column).

Row first, Offset—Specification of two format elements for the
cursor-address bytes must be entered here.  For the first element, a
numeric non-zero response to Row first element indicates that the row is
addressed by the first byte and the column is addressed by the second.  A
zero response indicates that the column is addressed by the first byte.
For the second element, the numeric response indicates the value of the
Offset used in both address bytes (this value is usually >20 to make the
addresses more easily "printable" in older high-level languages).

Definition of Optional Display-Control Information
Five other prompts list the definitions of display-control codes which can be
modified or accepted for default values. When a terminal or display does not
support features described in a prompt, the corresponding value must be set to
zero.  When a terminal or display can perform the functions specified in the
prompts, faster and more convenient operation results from entering the
codes.  The prompts, in the order in which the editor presents them, are
listed below.

Bell—The code used to activate the bell or beeper inside the terminal.
If a value of 0 is entered, the CC-40 uses the internal beeper.

Erase to end of line—The code used to erase all displayed characters from
the current cursor position to the end of the current line.

Erase to end of screen—The code used to erase all displayed characters
from the current cursor position to the end of the screen.

Insert line—The code used to enter a blank line on the screen.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  △ = percent sign;  σ = at sign;  μ = question mark

___

Tab stops--The column or character positions of the default tab stops.  Up
to ten column numbers can be entered in pairs separated by commas in
response to the five appearances of this prompt.


Default Values for Display-Control Characteristics
Default values for screen size and display-control codes are represented in
the following table.


## DEFAULTS FOR  DISPLAY-CONTROL CODES, DISPLAY SIZE, AND OPTIONAL CODES

| DISPLAY FUNCTION | EXTERNAL KEYBOARD (Televideo 920) | INTERNAL KEYBOARD (CC-40) |
|---|---|---|
| Screen size | 80,24 | 40,24 |
| Clear screen | ESC | 31 |
| Move cursor | ESC = | ESC = |
| Row first, | 1 (yes) | 1 (yes) |
|  offset | 32 | 0 |
| Bell | CTL G | 0 (CC-40 internal) |
| Erase to EOL | ESC T | ESC T |
| Erase to EOS | ESC Y | ESC Y |
| Insert line | ESC E | ESC E |
| Tab stops | 2,8,15,25 32,44,60,79 | 2,8,15,25,32 |


Using or Saving the Definition Table
After the response to the final Tab stops prompt is given, the screen editor
definition menu returns to the display. Pressing [Q] for Quit causes a return
to the main edit menu display for selection of the Screen option for testing
the table.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  ∂ = at sign:  u = question mark

After the table values are tested and the main edit menu reappears, pressing
[D] to select D)efine once more displays the screen-definition menu. Pressing
[S] for S)ave and entering a mass-storage device number and filename saves the
table in a file for later use.

## LINE AND SCREEN-EDITOR COMMAND DESCRIPTIONS

Editor commands, in general, perform identical functions in both line and
screen editing. Complete descriptions of these commands follow. Any
differences between command use in line editing and command use in screen
editing are noted in the descriptions. For clarity, only the default keys for
the terminal are listed: default keys for the CC-40 keyboard can be
referenced above in the table of keyboard default values. Syntax requirements
and options available for each command are summarized at the end of this
chapter.

### AUTO: Automatic Line Number Command

The AUTO command provides new line numbers automatically during line editing.
Followed by an optional line number (which defaults to 00001) in response to
the command cursor ">", AUTO displays the text line with the number entered.
If there is no line with that number, the line number and a trailing space are
displayed for entry of a new line at the end of text. If a line with the
number is already in the text, AUTO provides a line number followed by "+",
indicating that upon entry it is inserted into the text after the line of that
number. When [[ENTER]] is pressed, the line number is incremented by one, and
the next line number (followed by a space or "+", depending on whether the
line is to be appended or inserted) is displayed.

Initial entry into automatic line-number operation requires that the editor be
in command mode with the ">" being displayed. The word [AUTO] is typed,
followed by an optional line number and [[ENTER]]. Pressing [[BREAK]]
abandons the current display line without storing it in the text area of
memory and returns the editor to command mode with the ">" prompt being
displayed.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign:  & = percent sign:  c = at sign;  v = question mark

---

The forms of the AUTO command are shown in the following examples.


## AUTO: AUTOMATIC LINE NUMBERING

### LINE EDITING ONLY


AUTO            Begin auto operation with line 1
AUTO 100        Begin auto operation with line 100



## COPY: Copy (Load) a File into Text Area of Memory

The COPY command loads a file for editing from a mass-storage device, such as
a Wafertape™ device, into the text area of memory.

In line editing, if other text is already in memory and no line number is
included in the command, the file is loaded at the beginning of the text
area.  If a line number corresponding to a line in the text area of memory is
specified, the file is loaded (inserted) before the line in memory.  In screen
editing, the file is always loaded before the line containing the cursor.

If crunched mode has been specified in response to the main edit menu, then
the lines in memory contain command-representative tokens and suppressed
spaces.  Otherwise the lines are ASCII-character representations of the lines
originally entered.  If the file being copied into memory has been saved in a
mode different from the one currently in effect, the editor performs the
necessary conversion.

At the outset of COPY execution, when the editor attempts to open a file, an
error results in an Open error NN [file name],Retry: prompt.  Changes in media
are permitted before a [Y] response is given.  With a [N] response, the COPY
command is terminated.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  ц = question mark

If an error occurs while file lines are being loaded into memory or if there
is not enough space in memory to load the complete file, loading terminates
with the record being loaded when the error occurs.  An error message is
displayed.  All file lines successfully loaded into memory before the record
during which termination occurs are retained in memory for editing, saving, or
other operations.

The forms of the COPY command are shown in the following examples.


## COPY: LOAD FILE FROM MASS STORAGE UNIT INTO RAM


### LINE EDITING


COPY 1.XYZ             Load filenamed "XYZ" from device 1 into empty text area
                      of RAM


COPY 1.XYZ             Insert filenamed "XYZ" from device 1 into RAM before the
                      text now in RAM


COPY 1.XYZ 12         Insert filenamed "XYZ" from device 1 into the current
                      memory text before line 12


COPY 1.XYZ 1000       Append filenamed "XYZ" from device 1 after the current
                      memory text of less than 1000 lines


### SCREEN EDITING


[[ESC]] [C] 1.XYZ     Load filenamed "XYZ" from device 1 into text area of RAM
                      at line before cursor

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
#iona = pound sign:   & = percent sign;   @ = at sign:   ? = question mark

## DELETE: Delete Line or Block of Lines

The DELETE command removes specified lines from the text area of memory. A
single line is deleted when the command is followed by the number of the line
(e.g. DELETE 5). A range of lines to be deleted is specified by line numbers
separated by a hyphen. For example, DELETE 2-5 causes lines 2, 3, 4, and 5 to
be deleted.  The first line in the range defaults to 1 (DELETE -5 deletes
lines 1 through 5).  The last line number defaults to the last line in the
text (DELETE 1- removes all lines from the text).

When DELETE removes lines from the text area of memory, the deleted lines are
moved to another area of memory called the recover buffer.  The lines remain
in the recover buffer until some action is taken to change the text area of
RAM or until a MOVE, VERIFY, LIST, COPY, or SAVE command is issued.  The
contents of the recover buffer are lost if a line is modified, a line is
inserted, or a line is deleted.  If no action has been taken which alters the
contents of the recover buffer and an UNDO command is entered, the lines in
the buffer are inserted into the text once more.  The recover buffer and the
UNDO command minimize the possibility of accidentally destroying text.

A special form of DELETE causes the editor to remove all lines from the text
area of RAM without the delay necessary in first copying the text into the
recover buffer.  DELETE ALL performs very rapid text deletion, but it provides
for no subsequent recovery of the deleted text from the recover buffer.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  ∆ = percent sign;  ⌐ = at sign:  ⌐ = question mark

The forms of the DELETE command are shown in the following examples.

## DELETE: REMOVE A LINE OR A BLOCK OF LINES FROM THE TEXT

### LINE EDITING

DELETE 25          Delete line 25
DELETE 25-30       Delete lines 25 through 30
DELETE 25-         Delete line 25 and all following lines
DELETE -25         Delete all lines up to and including line 25
DELETE ALL         Delete text without copying to recover buffer

### SCREEN EDITING

[[ESC]] [D] 25       Delete line 25
[[ESC]] [D] 25-30    Delete lines 25 through 30
[[ESC]] [D] 25-      Delete line 25 and all following lines
[[ESC]] [D] -25      Delete all lines up to and including line 25
[[ESC]] [D] ALL      Delete text without copying to recover buffer

### EDIT: Display a Line for Editing

The EDIT command, used in line editing only, transfers a line from the text
area of memory into the display buffer. If the line editor has not been given
a NOLINE command to eliminate the display of line numbers, the display
contains a five-digit line number, a space, and the text of the line. If a
NOLINE command is in effect, the first column of the display contains the
first character of text in the line.

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
alpha = pound sign;  % = percent sign;  @ = at sign;  ? = question mark

The forms of the EDIT command are shown in the following examples.

## EDIT: DISPLAY LINE FOR EDITING

### LINE EDITING

EDIT 38           Display line 38 for line editing
E 38             Display line 38 for line editing

## FIND

The FIND command is used to display the next line in the text area of memory which contains a specified string (sequence) of characters. In both line and screen editing, the search for the string begins at the current line (with line editing) or the cursor (with screen editing) and continues until either (1) the string is found or (2) the end of the text is reached.

One option can be specified in the search: W. With the W option, no distinction is made in the search between uppercase and lowercase characters. Also, with the W option, the search is carried for an exact word match (including word boundaries); ABC does not match XABC or ABCY. For the string ABC to be matched by the identical sequence in the text, a string must be both preceded and followed by a nonword character. A nonword character is any character other than an ASCII alphabetic character, numeric character, or "$".

If the W option is not specified, alphabetic-character case is significant: ABC does not match AbC. The string is found in text regardless of the context in which it occurs or the delimiters surrounding it: ABC does match XABC and ABCY.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed: double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  a = at sign;  u = question mark

When a match occurs during the text search, the line in which the match occurs
is displayed and the cursor appears over the first character in the matched
string.

In the FIND command, any nonword character except "space" can be used to mark
the beginning and end of the string to be found.  Any printable character or a
space can be used inside the string, except for the character used as the
beginning and end marker.  The second occurrence of the marker signals the end
of the string.  In other words, the marker should be selected from among
punctuation characters which are not part of the string to be searched for.

In line editing, the FIND command is transferred to the playback buffer each
time [[ENTER]] is pressed.  For repeated searches for the same string,
pressing [[SHIFT]] [Up Arrow] transfers the FIND command back into the display
buffer for re-execution (subsequent search for the next occurrence of the
string in the text area of memory).  In screen editing, the string to be found
defaults to the string previously used: pressing the command initiation key
followed by [[ENTER]] continues the search.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alone = pound sign;  % = percent sign;  @ = at sign;  ⍰ = question mark

The forms of the FIND command are shown in the following examples.


### FIND: DISPLAY NEXT LINE CONTAINING STRING IN COMMAND


### LINE EDITING

FIND .XY.              Find "XY" in the current line or in a line following
                       it and display the string


F /X.Y/                Find "X.Y" in the current line or in a line following
                       it and display the string


FIND W .XY.            Find "XY", "Xy", "xY", or "xy" in the current or
                       following lines and display the string


F W .XY.               Find "XY", "Xy", "xY", or "xy" in the current or
                       following lines and display the string

### SCREEN EDITING


[[ESC]] [F] .XY.       Find "XY" in the current line or in a line following
                       it and display the string


[[ESC]] [F] W .XY.     Find "XY", "Xy", "xY", or "xy" in the current or
                       following lines and display the string



FORMAT: Prepare New Mass-Storage Medium

The FORMAT command followed by a mass-storage device number causes the media
in the device to be initialized for use. For example, FORMAT causes a
Wafertape device to write a blank "tape header" onto the tape currently in the
unit, effectively setting the tape up to receive files "saved" to it.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign:  u = question mark

The medium must be placed in the device before the command is issued.  All
files previously stored on the medium are erased.

The forms of the FORMAT command are shown in the following examples.

### FORMAT: INITIALIZE MEDIUM IN MASS-STORAGE DEVICE

#### LINE EDITING

FORMAT 2              Initialize new medium in device 2

#### SCREEN EDITING

[[ESC]] [N] 2     Initialize the new medium in device 2

### HELP: Show Commands and Display Functions

The HELP command, used in screen editing only, displays a list of the commands
and functions used in editing.  The commands are displayed one at a time.
While a command or function is being displayed, pressing keys on the keyboard
indicates the key sequence defined for it.  All key sequences except the
correct one cause the terminal or console to emit a beep.

Pressing [Up Arrow] displays the previously displayed command or function.
Pressing [Down Arrow] displays the subsequent one.  Pressing [[COMMAND EXIT]]
terminates the HELP display and returns the cursor to the text.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign; & = percent sign; @ = at sign; u = question mark

The form of the HELP command is shown in the following example.

## HELP: DISPLAY COMMANDS AND FUNCTION KEYS

### SCREEN EDITING

[[ESC]] [H]    Display commands and functions

JUMP: Move the Cursor (or Screen) to Another Point in the Text
The JUMP command causes the cursor to be placed in the line of the text which
corresponds to a line number specified in the command.  If the line is
currently being displayed on the screen, the cursor moves to the line without
any paging.  If the line is not on the screen, the display is paged so that
the specified line appears on the screen with the cursor.

The line number specified in the JUMP command can be written in two ways:
absolutely or relatively.  An absolute line number simply specifies the number
of the line to next appear with the cursor (e.g. "JUMP 10" causes the cursor
to move to line 10).  A relative line number (a number prefaced by [-] or [+])
specifies the number of lines before or after the current line that the line
to next appear with the cursor occurs (e.g., "JUMP +1" with the cursor
currently on line 10 causes the cursor to move to line 11, and "Jump -1"
causes it to move to line 9).

Two special forms of the JUMP command cause the cursor (and screen) to be
moved to the top (beginning) and bottom (end) of text.  The form "JUMP T"
causes the first page of text to be displayed with the cursor in the first
line; "JUMP B" causes the last page of text to be displayed with the cursor in
the last line.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;   Δ = percent sign;   σ = at sign;   u = question mark

If a relative line number calls for cursor placement in a line with a number
of less than 1 or greater than the number of the last line of the text, the
cursor appears, respectively, at the first line or last line of the text.

The four forms of the JUMP command are shown in the following examples.

## JUMP: MOVE THE CURSOR (AND SCREEN) TO ANOTHER LINE

### SCREEN EDITING

[[ESC]] [J] 100      Move the cursor to line 100

[[ESC]] [J] +3      If the cursor is on line 100, move it to line 103, if
it is on 5, move it to 8; and so on.

[[ESC]] [J] T      Move to the top (first) line of the text

[[ESC]] [J] 8      Move to the bottom (last) line of the text

## LIST: Display or Print Lines of Text

The LIST command causes specified lines in the text area of memory to be
displayed or printed. Device parameters for this command must be enclosed in
double quotes. If no line-number range is specified, the entire text is
listed. When no first or last line number is specified in the command, line
numbers default (respectively) to line 00001 and the last line in the text (as
in the DELETE command).

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  c = at sign;  y = question mark

In line-editing, the first line appears in the LCD display if the LIST command
contains no device parameters.  It can be edited as if the [E] command had
been pressed in response to the command-mode prompt ">".  Subsequent lines
appear sequentially each time [[ENTER]] is pressed.  If a device number of a
printer, for example, or a device number and filename for a mass-storage
device are specified in the LIST command, the file lines are printed or
transferred to the specified device or file rather than displayed.

Lines are printed either with or without numbers in accordance with the
LINE/NOLINE command in effect during line editing or the [[CTL]] [T] toggle in
effect during screen editing.

In screen editing, the LIST command must contain device parameters.  There is
no provision for listing to the screen device.  The device number must not
specify the terminal or display device being used for screen editing.

At the outset of LIST execution, when the editor attempts to open a device an
error results in an Open error NN [device],Retryu prompt.  With an [N]
response, the LIST command is terminated.  If an error occurs while file lines
are being output, listing terminates with the record being listed when the
error occurs.  An error message is displayed.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign:  u = question mark

The forms of the LIST command are shown in the following examples.


## LIST: DISPLAY OR PRINT LINES OF TEXT

### LINE EDITING

LIST                    List memory text line-by-line to display

LIST 250-               List text lines 250 and following

LIST -250               List text lines through line 250

LIST "20"               List text to device number 20

LIST "1.DATA"           List all lines to a file named "DATA" on
                        mass-storage device number 1

LIST "20" 50-           List lines 50 and following to device 20

LIST "1.A" -50          List lines through 50 to a file named "A" on device
                        1


### SCREEN EDITING

[[ESC]] [L] "20"        List text to device number 20

[[ESC]] [L] "1.DATA"    List all lines to a file named "DATA" on
                        mass-storage device number 1

[[ESC]] [L] "20" 50-    List lines 50 and following to device 20

[[ESC]] [L] "1.A" -50   List lines through 50 to a file named "A" on
                        mass-storage device 1

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  ∆ = percent sign;  σ = at sign;  υ = question mark

## MOVE: Move a Line or Block of Text

The MOVE command copies a specified line or group (block) of lines from one
point in the text area of memory to another and deletes the original lines.
The block to be moved is specified by line number as in the list and delete
commands: if no last address is specified for a block, only one line which
corresponds to the first line number is moved.  The line of the text before
which the line or block is to be moved is also specified by line number.  The
command MOVE 1-10,20 moves lines 1 through 10 to a position in the text just
before line 20.

The default for the line that begins the block of lines is 1, and the default
for the line that ends the block is the last line in the text.

The forms of the MOVE command are shown in the following examples.


## MOVE: MOVE LINE OR LINES TO ANOTHER POINT IN TEXT

### LINE EDITING

MOVE 3,6                    Move the third line so that it becomes the fifth

MOVE 1-10,21                Move the first 10 lines of the text so that they
                           become the second 10

### SCREEN EDITING

[[ESC]] [M] 3,6             Move the third line so that it becomes the fifth

[[ESC]] [M] 1-10,21         Move the first 10 lines of the text so that they
                           become the second 10

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
alpha = pound sign; Δ = percent sign; σ = at sign; u = question mark

## QUIT: Return to the ALDS Menu

The QUIT command causes the CC-40 to exit the editor and return to the ALDS menu. Before the editor is exited, however, the prompt Are you sure (Y/N)u appears in the console display or in the last line of the screen display. Pressing [Y] for YES permits the exit to occur. Pressing [N], causes a return to the editor.

Once [Y] is pressed in response to the prompt, all text in memory is lost.

The forms of the QUIT command are shown in the following examples.

## QUIT: RETURN TO EDIT MENU

### LINE EDITING

QUIT                      Leave line editing and return to edit menu

### SCREEN EDITING

[[ESC]] [Q]               Leave screen editing and return to edit menu

## REPLACE: Find One String and Replace It with Another

The REPLACE command searches for the next occurrence of a specified string (sequence of characters) in the text area of memory and replaces it with another specified string. All alphabetic and numeric characters and spaces are significant in the search. There are three options which can be, either alone or in combination, selected with the REPLACE command: the word W option, the verify V option, and the all A option.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  % = percent sign;  σ = at sign;  υ = question mark
_____

The [W] option following a REPLACE command indicates that the search is for a
"word" pounded by nonword characters.  Individual characters of the word can
be either lowercase or uppercase.  The W option in the REPLACE command
operates just as in the FIND command.

With V option in effect when a string is found during line editing, the
display shows 19 characters of the line beginning with the string.  Following
the string is the prompt Replaceυ  A [Y] response to the prompt causes
replacement to occur; an [N] response causes the string to be skipped over
without replacement.  While the editor awaits a (Y/N) response, a line can be
scrolled right or left within the 19-character window by the use of [Left
Arrow] and [Right Arrow].  If option A is in effect, the search is continued;
if not, the search is terminated.

With the verify option in effect when a string is found during screen editing,
the cursor is placed over the first character of the string.  The prompt
Replace: Y)es N)o S)top appears in the command line.  A [Y] response to the
prompt causes replacement to occur; an [N] response causes replacement to be
skipped over.  A [S] response terminates the REPLACE command.

The REPLACE command followed by [A] indicates that all occurrences of the
searched-for string in the text are to be replaced.  When the verify option is
also specified, the replacement operation stops each time the searched-for
string is found to query about replacement.  After replacement of the string
is effected or skipped over by a [Y] or an [N] response, the search continues
through subsequent lines.  If a [S] response instead of a yes-or-no response
is given to the prompt, no replacement occurs and the REPLACE ALL command is
terminated.  If the verify option is not specified in a REPLACE ALL command,
every occurrence of the searched-for string is replaced.  The display reports
the number of times that replacement has occurred with the message Number of
replacements = NN when the ALL option is in effect.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

When a string is replaced, the original text of the line is saved in the
recover buffer.  If, before the recover buffer is overwritten in another
operation (such as another replace operation, line modification, or line
deletion), [UNDO] is pressed, the last replacement is voided: the original
text line is restored.

If the string which is to replace the searched-for string is null (that is, no
characters are typed between delimiters), the effect of the replacement is to
delete the searched-for string from the text area of memory.

The forms of the REPLACE command are shown in the following examples.


## REPLACE: FIND ONE STRING AND REPLACE IT WITH ANOTHER


### LINE EDITING


REPLACE .XX.YY.            Replace the next occurrence of "XX" with "YY".


REPLACE W uX.XuXXu         Replace the next occurrence of "X.X", "X.x",
                           "x.X", or "x.x" with "XX"


REPLACE V ...X.YY.         Replace the next occurrence of "XX" with "YY"
                           after displaying the line, prompting with
                           "Replaceu", and receiving a [y] or [Y] response


REPLACE A .XX.YY.          Replace all subsequent occurrences of "XX" with
                           "YY"


R A .XX.YY.                Same as "REPLACE A .XX.YY."


R AWV .XX.YY.              Replace as above with all options specified

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  % = percent sign;  @ = at sign;  ? = question mark

---

## SCREEN EDITING

[[ESC]] [R] .XX.YY.          Replace the next occurrence of "XX" with "YY"

[[ESC]] [R] W .XX.YY.        Replace the next occurrence of "XX", "Xx", "xX",
                             or "xx" with "YY"

[[ESC]] [R] V .XX.YY.        Replace the next occurrence of "XX" with "YY"
                             after placing the cursor at the string, prompting
                             with "Replace: Y)es N)o S)top", and receiving a
                             [y] or [Y] response

[[ESC]] [R] A .XX.YY.        Replace all subsequent occurrences of "XX" with
                             "YY"

[[ESC]] [R] AWV .XX.YY.      Replace as above with all options specified

---

## SAVE: Write the Text to a Mass-Storage File

The SAVE command causes the text in memory to be transferred to a file on a
mass-storage device.  The text is saved with lines separated from each other
by single-byte nulls.  If the text is in crunched format, it is saved to the
file in crunched format.  Line numbers are not saved.

The SAVE command permits optional specification of record size.  If no record
size is specified, the text is saved as one record.  If a size of less than 80
bytes is specified, the text is saved in 80-byte records.  If a size greater
than 80 is specified (1024, for example), records written during SAVE command
execution are of that length.

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
alpha = pound sign; Δ = percent sign; ɤ = at sign; ʮ = question mark

At the outset of SAVE execution, when the editor attempts to open a file, an error results in an Open error NN [device].Retryʮ prompt. Changes in media are permitted before a [Y] response is given. With an [N] response, the SAVE command is terminated. If an error occurs while text is being written to the file, the output operation terminates with the record being listed when the error occurs. An error message is displayed.

The forms of the SAVE command are shown in the following examples.

### SAVE: SAVE THE TEXT IN FILE ON MASS-STORAGE DEVICE

#### LINE EDITING

SAVE 2.TABLE

Save the current memory text to device 2 with the filename "TABLE" in a single record

SAVE 2.TABLE R=70

Save "TABLE" to device 2 in 80 character records

SAVE 2.TABLE R=>FF

Save the text as "TABLE" to device 2 with a record size of >FF (255)

#### SCREEN EDITING

[[ESC]] [S] 2.TABLE

Save the current memory text to device 2 with the filename "TABLE" a single record

[[ESC]] [S] 2.TABLE R=70

Save "TABLE" to device 2 in 80 character records

[[ESC]] [S] 2.TABLE R=>FF

Save the text as "TABLE" on device 2 with a record size of >FF (255)

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign:  b = percent sign;  c = at sign;  u = question mark
_____

## TAB DEFINE: Set Display Tab Stops Other Than as Defined

The TAB DEFINE command, used only in screen editing, redefines video-display
tab stops during editing.  The column positions of up to 10 stops, separated
from each other by commas, are entered on the last line of the display before
[[ENTER]] is pressed.  (The leftmost column is column 0, and the rightmost is
column 79.)

The form of the TAB DEFINE command is shown in the following example.

## SET TABS: TO POSITIONS OTHER THAN THOSE SET IN DEFINE

## SCREEN EDITING

   [[ESC]] [T] 5,10,15,20,25,30,35,40    Set tabs after every five columns

## UNDO: Cancel the Last Modification, Deletion, or Replacement

The UNDO command transfers a line or block of lines from the recover buffer
into the text area of memory.  The recover buffer always contains the most
recently deleted line or block or the most recently modified line of the
text.  Through UNDO, deleted lines are reinserted into the text or a modified
line is replaced by the original text of the line.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
aioha = pound sign;  Δ = percent sign;  ? = at sign:  u = question mark

The forms of the UNDO command are shown in the following examples.

## UNDO: RESTORE TEXT TO STATE BEFORE MODIFICATION OR DELETION

### LINE EDITING

UNDO            After line modification, replace the modified line with the
                original; after deletion, restore deleted lines to the text

### SCREEN EDITING

[[ESC]] [U]     After line modification, replace the modified line with the
                original; after deletion, restore deleted lines to the text

## VERIFY: Compare the Text with a Mass-Storage File

The VERIFY command compares character-for-character the text in memory with a
specified file on a mass-storage device.  If one or more of the respective
characters in the text and the file differ, the message I/O error 24, filename
is displayed.  (Other errors, such as file-not-found and device errors, can
also occur during a verify operation).  The primary use of VERIFY is to ensure
that file transfer between mass-storage and memory (through COPY or SAVE
commands) introduces no errors.

No record size is specified in the verify command.

underlining in text (except heads) indicates a typestyle other than normal
text: single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign:  s = percent sign;  c = at sign:  u = question mark

## TAB DEFINE: Set Display Tab Stops Other Than as Defined

The TAB DEFINE command, used only in screen editing, redefines video-display
tab stops during editing.  The column positions of up to 10 stops, separated
from each other by commas, are entered on the last line of the display before
[[ENTER]] is pressed.  (The leftmost column is column 0, and the rightmost is
column 79.)

The form of the TAB DEFINE command is shown in the following example.


## SET TABS: TO POSITIONS OTHER THAN THOSE SET IN DEFINE


## SCREEN EDITING


   [[ESC]] [T] 5,10,15,20,25,30,35,40    Set tabs after every five columns


## UNDO: Cancel the Last Modification, Deletion, or Replacement

The UNDO command transfers a line or block of lines from the recover buffer
into the text area of memory.  The recover buffer always contains the most
recently deleted line or block or the most recently modified line of the
text.  Through UNDO, deleted lines are reinserted into the text or a modified
line is replaced by the original text of the line.

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
alpha = pound sign;   Δ = percent sign;   z = at sign;   u = question mark

The forms of the UNDO command are shown in the following examples.

## UNDO: RESTORE TEXT TO STATE BEFORE MODIFICATION OR DELETION

### LINE EDITING

UNDO             After line modification, replace the modified line with the original; after deletion, restore deleted lines to the text

### SCREEN EDITING

[[ESC]] [U]      After line modification, replace the modified line with the original; after deletion, restore deleted lines to the text

## VERIFY: Compare the Text with a Mass-Storage File

The VERIFY command compares character-for-character the text in memory with a specified file on a mass-storage device.  If one or more of the respective characters in the text and the file differ, the message I/O error 24, filename is displayed.  (Other errors, such as file-not-found and device errors, can also occur during a verify operation).  The primary use of VERIFY is to ensure that file transfer between mass-storage and memory (through COPY or SAVE commands) introduces no errors.

No record size is specified in the verify command.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  c = at sign:  u = question mark

The forms of the VERIFY command are shown in the following examples.

## VERIFY: COMPARE MEMORY AND MASS-STORAGE FILES

### LINE EDITING

VERIFY 1.ABC                Compare the memory text with the filenamed "ABC"
                            on device 1; issue message if text and the file
                            are different

### SCREEN EDITING

[[ESC]] [V] 1.ABC           Compare the memory text with the filenamed "ABC"
                            on device 1; issue message if text and the file
                            are different

SUMMARY COMMAND DESCRIPTIONS

The following tables list summary command descriptions for line editing.

LINE EDITING TABLE

SCREEN EDITING TABLE

## LINE EDITING COMMAND SUMMARY

| Command | Description |
|---|---|
| AUTO<br>AUTO 100 | Begin automatic line numbering with line 1<br>Begin automatic line numbering with line 100, or if text contains line 100 begin insertion |
| COPY 3,XYZ<br><br>COPY 3,XYZ 10<br><br>COPY 10,XYZ 1000 | Load or insert file "XYZ" from device 3 into RAM at the beginning of the text area<br>Insert file named "XYZ" from device 3 into the current memory text before line 10<br>Append file named "XYZ" from device 10 to current text of less than 1000 lines |
| DELETE 25<br>DELETE 25-30<br>DELETE 25-<br>DELETE -25<br>DELETE ALL | Delete line 25<br>Delete lines 25 through 30<br>Delete line 25 and all following lines<br>Delete all lines up to and including line 25<br>Delete text without copying to recover buffer |
| EDIT 38 | Display line 38 for line editing |
| FIND ,XX,<br>FIND W ,XX, | Find the next occurrence of "XX" and display it<br>Find the next occurrence of "XX", "Xx", "xX", or "xx" and display it |
| FORMAT 7 | Initialize the media in device 7 |
| LINE ENTRY COMMANDS<br>100 This[ENTER]<br><br><br><br>100+This[ENTER]<br>100-This[ENTER]<br>100 This[↑]<br><br>100 This[↓] | With ">" showing, enter the string "This" into text line 100 or replace the line currently at position 100 of the text with the string "This"<br>Enter (insert) "This" into new line 101<br>Enter (insert) "This" into new line 99<br>While editing, enter "This" as line 100 and thereafter display line 99<br>While editing, enter "This" as line 100 and thereafter display line 101 |
| LIST<br>LIST 250-<br>LIST -250<br>LIST "20"<br>LIST "5,DATA"<br><br>LIST "20" 50-<br>LIST "5,A" -50 | List memory text line-by-line to display<br>List text lines 250 and following<br>List text lines through line 250<br>List text to device number 20<br>List lines to a file named "DATA" on mass-storage device 5<br>List lines 50 and following to device 20<br>List lines through 50 in file named "A" on mass-storage device 5 |
| MOVE 1-10,21<br><br>MOVE 20-,1 | Move the first 10 lines of the text so that they become the second ten<br>Move lines 20 and following so that they become the first block of lines in the text |
| QUIT | Leave line editing and return to ALDS menu |
| REPLACE ,XX,YY,<br>REPLACE W ,XX,YY,<br><br>REPLACE V ,XX,YY,<br><br>REPLACE A ,XX,YY,<br><br>REPLACE AWV ,XX,YY, | Replace the next occurrence of "XX" with "YY"<br>Replace the next occurrence of "XX", "Xx", "xX", or "xx" with "YY"<br>Replace the next occurrence of "XX" with "YY" after verifying<br>Replace all subsequent occurrences of "XX" with "YY"<br>Replace as above with all three options in combination |
| SAVE 2,TABLE<br><br>SAVE 2,TABLE R=0<br><br>SAVE 2,TABLE R=>FF | Save the current memory text to device 2 with the file name "TABLE" in a single record<br>Save "TABLE" to device 5 in record size of 90<br>Save the text as "TABLE" on device 2 with a record size of 255 |
| UNDO | After line modification, replace the modified line with the original or after deletion restore deleted lines to the text |
| VERIFY 1,ABC | Compare the memory text with the file named "ABC" on device 1; issue message if the text and the file are different |

## SCREEN EDITING COMMAND SUMMARY

| Command | Description |
|---|---|
| [ESC][C] 1.XYZ | Load file named "XYZ" from device 1 into empty text area of RAM |
| [ESC][C] 1.XYZ | Insert file XYZ before line containing cursor |
| [ESC][D] 25 | Delete line 25 |
| [ESC][D] 25-30 | Delete lines 25 through 30 |
| [ESC][D] 25- | Delete line 25 and all following lines |
| [ESC][D] -25 | Delete all lines up to and including line 25 |
| [ESC][D] ALL | Delete text without coping to recover buffer |
| [ESC][F] .XX. | Find the next occurrence of "XX" and place the cursor after it |
| [ESC][F] W .XX. | Find the next occurrence of "XX", "Xx", "xX", or "xx" and place the cursor after it |
| [ESC][H] | Display editor commands and functions |
| [ESC][J] 100 | Jump the cursor to line 100 |
| [ESC][J] +3 | If the cursor is on line 100 jump it to line 103, if it is on 5 jump it to 8, etc. |
| [ESC][L] "20" | List text to device number 20 |
| [ESC][L] "6.DATA" | List lines to a text file named "DATA" on mass-storage device number 6 |
| [ESC][L] "21" 50- | List lines 50 and following to device 21 |
| [ESC][L] "6.A" -50 | List lines through 50 to a file named "A" on mass-storage device 6 |
| [ESC][M] 1-10,21 | Move the first 10 lines of text so that they become the second ten |
| [ESC][M] 20,1 | Move lines 20 and following so that they become the first block of lines |
| [ESC][M] 1-10,20 | Move the first 10 lines of text so that they become the second ten |
| [ESC][M] 20,1 | Move line 20 so that it becomes line 1 |
| [ESC][N] 3 | Initialize (format) the medium in device 3 |
| [ESC][O] | Leave screen editing and return to ALDS menu |
| [ESC][R] .XX.YY. | Replace the next occurrence of "XX" with "YY" |
| [ESC][R] W .XX.YY. | Replace the next occurrence of "XX", "Xx", "xX", or "xx" with "YY" |
| [ESC][R] V .XX.YY. | Replace the next occurrence of "XX" with "YY" after verification |
| [ESC][R] A .XX.YY. | Replace all subsequent occurrences of "XX" with "YY" |
| [ESC][R] AWV .XX.YY. | Replace as above with all three options |
| [ESC][S] 5.TABLE R=0 | Save the text currently in memory to device 5 the file name "TABLE" *IN A FILE WITH 50 Bytes RECORDS* |
| [ESC][S] 5.TABLE R=>FF | Save the text as "TABLE" on device 5 with a record size of 255 |
| [ESC][S] 5.TABLE | Save "TABLE" to device 5 in one record |
| [ESC][T] 5,10,15,20,25,30,35,40 | Set tabs after every five columns |
| [ESC][U] | After line modification, replace the modified line with the original; after deletion restore deleted lines to the text. |
| [ESC][V] 1.ABC | Compare the memory text with the file named "ABC" on device 1; issue message if the text and the file are different. |

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign:  & = percent sign;  c = at sign;  u = question mark

---

## CHAPTER 5
## THE ASSEMBLER

This chapter describes the way the assembler is used to produce object files
from source-file text.  It also describes, in overview and in detailed
discussion, the assembler instructions called pseudo operations and
directives, and it discusses assembly errors.

The assembler is an ALDS program which converts an assembly-language source
program stored on a mass-storage device such as a Wafertape™ device into a
machine-language program.  It translates the easily remembered (mnemonic)
expressions for processor commands, program addresses, and program-related
values (source code) into machine language (object code).  The object code,
when suitably linked with other programs and loaded into CC-40 system RAM or
CC-40 cartridge RAM, can be executed by the TMS7000-family processor.

## SOURCE AND OBJECT FILES

The assembler source file is an assembly-language text file which has been
written to a mass-storage device such as a Wafertape™ device.  The object
file, also written to a mass-storage device, contains machine-language
commands, references to variables in other programs, and information required
for linking and running TMS7000-family machine-language programs.

### Source Files

The assembly-language program, stored as a file on a mass-storage device by
the ALDS editor, can be in either crunched or uncrunched format.  In crunched
format, keywords (assembly-language op codes and assembler instructions) are
represented by single-byte tokens and multiple spaces are suppressed.
Crunched format thus saves storage space.  The input file can be stored as
either a single- or multiple-record file as specified in the editor SAVE
command.  Single null (>00) bytes separate the lines of text in the file.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed

alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

## Object Files

The assembler output (object) file consists of an absolute or relocatable
machine-language program which can subsequently be processed for execution
from CC-40 memory.  The object file can be saved on a mass-storage device in a
single record or in multiple records of a specified length. Relocatable object
code flags address values in object code produced by direct addressing
commands such as LDA, STA, CALL, and BR.  These addresses are subsequently
evaluated by the linker relative to the start of the machine-language program.

The format of the object-code program is tagged-object format.  In this
format, assembly-language labels which have been converted into absolute or
relocatable values within a single object file can be resolved from other
tagged-object files being linked with it for execution.  Linking resolves
references external to the assembler-output file by transferring labeled
addressing and value information among files to be linked.

Tagged object format stores object code in a series of variable-length
fields.  Each field begins with a single alphabetic label or tag.  The label
or tag for each field uniquely identifies the format, the contents, and the
type of data in the field.  A complete description of CC-40 tagged-object code
appears in Appendix E of the CC-40 Editor/Assembler Reference Manual.

## RUNNING THE ASSEMBLER

After the ALDS is entered from BASIC with a [RUN "ALDS"] command, the ALDS
program menu appears in the display as shown below.

### E)dit A)ssemble L)ink B)asicu

The assembler is entered when [A] is pressed.  The assembler displays a series
of prompts which request the names of mass-storage files, and the designations
of peripheral devices which are to be used during assembly.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alone = pound sign:  % = percent sign:  @ = at sign:  u = question mark

Upon entry, the assembler prompts for (1) source file, (2) object file, (3)
type of assembly, and (4) listing device number or filename. These prompts,
along with typical responses, are shown below.


    Source file [1.SRC]
    Object file [2.OBJ R=1024]
    In memory assemble (Y/N)u [N]
    List file [21.B=300,R=N]


The object file is optional. No object file is produced if [[ENTER]] is
pressed. Assembly without an object file is useful for displaying or printing
assembly errors in minimal time.


When an object-filename is entered, it can be followed by spaces and a
specification for record length (of the form R=nnn, with nnn being the number
of bytes in a record). Specifying no record length with the filename
indicates that the object file is to be written as a single record). In the
example above, the object file is given the name "OBJ"; the file is written to
device 2 in 1024-byte records.


The In memory assembleu prompt does not always appear: it appears only if an
object file is to be provided by assembly. An answer to the prompt depends on
the length of text being assembled and CC-40 system configuration. An [N]
answer to the prompt, when mass-storage devices such as Wafertape™ units
(which permit only one file to be open at a time) are used, requires that
separate devices be used for source file and object file. A [Y] answer
indicates that the object code is stored in memory until the source file can
be closed and the object file written to the storage device. For an in-memory
assemble, memory must be large enough to contain source code, object code, and
assembly tables at the same time.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

Much larger files can be assembled when an [N] answer is given to the In
memory assembled prompt than when a [Y] answer is given, but the mass-storage
device configuration of the CC-40 system must support two files being open
simultaneously, either with two devices such as Wafertape$^{TM}$ devices  or with
some other single device which permits multiple open files.

If no listing device or filename is entered, no listing is produced and
assembly errors are displayed on the console.  When the List file prompt is
answered with a device number, as with the RS232 device specified in the
example above, the listing can be printed or displayed on an external device.

## SOURCE FILE STATEMENTS
The assembler source file contains three kinds of single-line statements: (1)
assembly-language instructions for the assembler to translate into machine
commands, (2) pseudo-operation instructions (called "pseudo-ops" for short)
which result in storage of data into the object-code output of the assembler,
and (3) assembler directives.

## Assembly-Language Instructions
Assembly-language instructions consist of mnemonic (memory-aiding) labels and
op codes, mnemonic or numeric operands, and comments (see chapter 2 of this
manual and chapter 6 of the CC-40 Editor/Assembler Reference Manual for more
detailed information).  The assembler translates these instructions
line-by-line into machine-language commands in the object code.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  % = percent sign;  ? = at sign;  u = question mark

## Pseudo Operations

Pseudo-operation instructions direct the assembler to store data other than
processor commands into object code.  Pseudo-ops are placed in the op-code
field of source lines, and the data is placed in the operand field.  This data
can consist of binary bytes or words for use in data tables, strings of ASCII
characters to be used as messages, applications-program menu keywords, or
other such data.  The assembler recognizes two pseudo-ops for writing numeric
data into the object file and two pseudo-ops for writing ASCII
(character-string) data into the object file, as described below.  More
detailed information about pseudo-ops is given in the following discussion.


## Numeric-Data Pseudo Operations

Two instructions cause the assembler to store numeric data into the object
file.  One is for storing single-byte values in the range from 0 through 255,
and the other is for storing double-byte ("word") values in the range from 0
through 65535.  The instructions are as follows.

   BYTE:   Write one or more single-byte values specified by the operand into
           the object file.

   DATA:   Write one or more two-byte (word) values specified by the operand
           into the object file.


## ASCII (Character-String) Data-Storage Pseudo-Ops

Two instructions cause the assembler to store bytes representative of ASCII
text characters into the object file.  One instruction stores characters of a
string in "normal order" and the other instruction stores characters in
"reverse order."  These two string pseudo-ops are as follows.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  μ = question mark

TEXT:   Write the ASCII representations of characters in the string
        specified by the operand into the object file in normal order
        (first character at the lowest object-file address, last character
        at highest object-file address).  A label preceding the TEXT
        instruction is evaluated to the address of the first byte of the
        string (lowest address).

RTEXT:  Write the ASCII representations of characters in the string
        specified by the operand into the object file in reverse order
        (last character at lowest object-file address, first character at
        highest object-file address.  This instruction is particularly
        useful because the TMS7000-family processor has a memory-pointer
        decrementing instruction (DECD) but no pointer incrementing
        instruction.  A label preceding the RTEXT instruction is evaluated
        to the address of the last byte of the string (highest address).

## Assembler Directives

Assembler directives specify the manner in which program assembly is carried
out. Although they do not produce object code, these instructions are also
typed into the op-code field, and any values used with them are placed in the
operand field.

Assembler directives cause, for example, assembly of object code for use at
particular locations in memory (relative or absolute), and they cause listing
or suppression of listing of assembly source and object code.

The assembler directives include the following groups of instructions.

    !o! Four instructions for specifying the location at which subsequent code
        and data are to be written into the object file

    !o! An instruction for equating symbols with values during assembly

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
aioht = pound sign;  x = percent sign;  o = at sign;  u = question mark

─────────────────────────────────────────────────────────────────────────

!o! An instruction for specifying the end of code for a program

!o! Two instructions for producing object-code symbols which enable
    multiple object-file linking

!o! An instruction for including another source file within the current
    one during assembly

!o! Five instructions for printing and formatting assembly-output listing
    pages

!o! An instruction for setting assembler options, such as producing
    cross-reference (symbol/location) listings and truncating the listing
    of lines of object code for byte, data, and text lines

The instructions within each group of directives are described below.


Code and Data Location Instructions

Four instructions control program memory locations in which code and data can
be stored during program execution. Two instructions directly set the
assembler "program counter," and the other two cause the assembler program
counter to increment past memory locations where data is to be stored.

  AORG:    Absolute origin: Set the absolute-address origin of subsequent
           object code to the value specified in the operand.

  RORG:    Relocatable origin: Set the relative-address origin of the object
           code to an address displaced from relative address 0 by the value
           specified in the operand.

  BSS:     Block starting with symbol: Skip past object-code locations for
           the number of bytes specified in the operand and assign any label
           for the skipped bytes to the address of the first byte skipped.

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
alpha = pound sign; & = percent sign; σ = at sign; ν = question mark

BES:    Block ending with symbol: Skip past object-code locations for the number of bytes specified in the operand and assign any label for the skipped bytes to the address of the first byte after the block skipped.

## Symbol Value Equating Directive

The EQU directive assigns values to symbols which precede it in the label column of the line.

EQU:    Equate: Assign the value in the operand to the label.

## End of Source Statement

The END directive marks the end of the assembler source code. Any source-code instructions following an END directive are ignored by the assembler.

END:    End source code: Terminate assembly at this line.

## Linking Information Directives

Two directives provide the ALDS linker with information making program variables accessible to linked programs and permit the use of similar variables in the other programs. (Also, see "IDT," below.)

DEF:    Define symbol: Include the symbol specified in the operand as a "public" symbol in the object file with a value made available to the linker for use in other programs being linked to the current one.

REF:    Reference symbol: Include the symbol specified in the operand as an "external" symbol in the object file with a value to be searched for by the linker among other programs being linked to the current one.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  % = percent sign;  @ = at sign;  ? = question mark

## Source File Copy (Include) Directive

The COPY directive enables the assembler to produce object files from multiple
source files.

COPY:    Copy file: Copy or include another source file named in the
         operand into the current file during assembly only (source files
         remain separate following assembly).

## Assembly-Listing Control Directives

Five directives control the listing of the assembly (and identify the program
in the object file for access by the linker).

IDT:     Identify program: Identify the program  by the string in the
         operand both for the listing page heading and for access by the
         linker.

TITL:    Title the following pages: Provide a page title for the next page
         and subsequent pages from the string in the operand.

LIST:    List lines: List the lines of the assembled program following this
         directive and set the number of lines to be printed on each page
         prior to automatic end-of-page eject.

PAGE:    Eject page: Eject a page in listing.

UNL:     Unlist lines: Do not list the lines of assembled program following
         this directive.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  ɑ = at sign;  ɥ = question mark

## Assembler Option Directive

The OPTION directive turns on and off the listing of multiple lines of object
code following DATA and TEXT instructions, and it turns on and off the listing
of a cross-reference map of assembly variables. When OPTION is placed in a
line in the position of an op code, the following option specifications
(separated by commas if more than one is required) can be listed in the
operand field.

B:  Truncate byte lines: Print only the first line of object code
    generated by each BYTE pseudo-op.

D:  Truncate data lines: Print only the first line of object code
    generated by each DATA pseudo-op.

T:  Truncate text lines: Print only the first line of object code
    generated by each TEXT pseudo op or the last line of each RTEXT
    pseudo-op.

F:  Finish line truncation: Print full object code generated by BYTE,
    DATA, TEXT, and RTEXT pseudo-ops.

X:  Produce a Cross-reference table: Produce a detailed cross-reference
    table listing the value of each symbol, the source-file line on which
    the value definition occurs, and the source-file lines containing
    references to the symbol (in operands).

R:  Produce a Reduced cross-reference table: Produce a cross-reference as
    above, except that only symbols from copied files which are referenced
    are listed in the table.

## Pseudo Operations in Detail

All pseudo operations which produce binary data in the object code are written
into source lines in the op-code field. The expressions to be evaluated into
binary data are written into the operand field of the line.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  o = at sign;  v = question mark

Each instruction can be preceded by a label which, for BYTE, DATA, and TEXT,
is evaluated during assembly to the location at which the first byte
representing the operand is stored.  A label for the RTEXT instruction,
however, is evaluated to the location at which the last byte representing the
operand is stored.

Numeric Data-Storage Instructions: BYTE and DATA
The BYTE instruction causes one or more single-byte values specified by the
operand to be written into the object file.  If one byte is to be written, the
instruction has the following form.

       BYTE >D

Multiple bytes can be written with a single instruction by entering multiple
values separated by commas into the operand field of the line, as in the
following.

       BYTE 13,>D

The DATA instruction causes one or more two-byte values (words) specified by
the operand to be written into the object file.  If one word is to be written,
the instruction has the following form.

       DATA >D

Multiple words can be written with a single instruction by entering multiple
values separated by commas into the operand field of the line, as in the
following.

       DATA 1023,-1023

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

The assembler evaluates negative values into binary form according to two's
complement convention.  The hexadecimal representation for the evaluation of
-1 is >FFFF, for example, and the hexadecimal representation of -65535 is
>0001.

String characters can be used in expressions in BYTE or DATA instruction
operands.  The instruction BYTE 'A'->40 writes >01 or CTL-A into the object
file.  Similarly, the instruction DATA 'AB' writes >4142 into the object file.

ASCII Data-Storage Instructions: TEXT AND RTEXT
The TEXT instruction causes a quoted string of ASCII characters specified in
the operand to be written into the object file in normal order: the first
character in the operand is written at the lowest object-file location, and
the last character is written at the highest object-file address.  If the
initial quotation mark in the operand is preceded by [-], the highest bit (bit
7) of the last byte of text stored is set to ONE to mark the end of the string.

The RTEXT instruction, like the TEXT instruction, causes a quoted string of
ASCII characters specified in the operand to be written into the object file.
The order in which the characters are written, however, differs with the RTEXT
instruction: the last character in the operand is written at the lowest
object-file location, and the first character is written at the highest
object-file address.  A label used with RTEXT is assigned the value of the
location of the first character (at the highest location).  Decrementing a
memory pointer set to this label with a DECD instruction during program
execution scans through the reversed text in normal order.  If the initial
quotation mark in the operand is preceded by [-], the highest bit (bit 7) of
the lowest-addressed byte of text stored is set to ONE to mark the end of the
string.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  ν = question mark

## Assembler Directives in Detail

Assembler directives can be divided into several classifications. Each of
these classifications is discussed below.

Code- and Data-Location Instructions: AORG, RORG, BSS, and BES
When translating assembly-language instructions and data into object code, the
assembler progressively maintains a two-byte counter which points to the
location of each byte of a machine command or data block in the object code.
As each byte of the object code is written, the pointer is incremented to the
next location in memory.

In general, when the assembler reads a label in the source code (during "pass
1," the initial assembler scan of the source code), it stores the label and
the value of the corresponding location counter in a table called the symbol
table. After the table is complete, the assembler scans the source code again
(during "pass 2" in which the object code is written). During pass 2, when a
symbolic (non-numeric) operand is found, the assembler searches the symbol
table in order to find a numeric value corresponding to the operand. The
numeric value from the table is then incorporated into the object code for the
assembly-language or data instruction currently being processed.

Four assembler directives provide control over the assembler location counter
from statements in the source program. The AORG (absolute origin) and RORG
(relocatable origin) instructions determine both the value of the location
counter and the action taken by the linker in processing the object code for
execution at specific memory addresses. The AORG and RORG instructions are
mutually exclusive during the assembly of any program. Assembly set by AORG
to produce absolute code cannot also produce relocatable code. Likewise,
relocatable-code assembly set by RORG cannot produce absolute code. The
assembler produces a Mixed module error message and disregards any origin
statement that is not of the same type as the first origin statement.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

When the AORG instruction is encountered, the two-byte location counter is set
immediately to the value specified in the operand. Subsequent
assembly-language and data instructions increment the counter from this
value. If this value is >4567, for example, then any label on the AORG
instruction itself or on the instruction following it is stored in the
assembler symbol table along with the value. If the label appears in an
instruction operand, the assembler uses >4567 appropriately in the object code
for the instruction during pass 2 of the assembly.

The RORG instruction performs essentially the same task as the AORG
instruction, except that it flags the object code following it for subsequent
relocation by the linker. The linker subsequently "biases" address-related
words in the object code following an RORG instruction by the address at which
the object code is later to be loaded. An RORG instruction assigning the
location counter a value >567, when the linker is set to relocate the origin
of the program to location >4000, results in an address of >4567 for the next
instruction.

If no AORG instruction occurs before the first object-code producing
instruction or memory-block skip instruction in a program (see BES and BSS
below), the origin for the program defaults to RORG 0. If an AORG directive
is subsequently encountered, a Mixed module error occurs.

The object code in a program can be stored discontinuously: the assembler
location counter can be incremented past a specified number of addresses to
reserve the skipped block of memory for data storage during processing. Two
directives cause such blocks of memory to be skipped: BSS (block skipped
defined by starting symbol) and BES (block skipped defined by ending symbol).
The operand of each instruction specifies the number of bytes added to the
location counter during both passes of assembly.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  ¢ = at sign;  υ = question mark

The only difference between the BSS and the BES instructions lies in the value
which the assembler assigns to a label included with the instruction during
first pass.  With a BSS instruction, the value assigned is the value in the
location counter when the instruction is encountered in the source code.  With
a BES instruction, the value assigned is that of the location counter after
the length of the block skipped has been added to it. The value is the address
of the next byte of object code to be stored.


Value-Assignment Instruction: EQU

When the assembler encounters an equate instruction in the source code during
the first assembly pass, it stores the value specified in the operand of the
instruction in the symbol table as the value assigned to the label of the
instruction.  If the operand is itself a symbol, it must be a symbol
previously evaluated to a number and stored in the symbol table.


Registers can be given mnemonic names through the use of equate statements.
C EQU R2, for example, can enable programming with another register designated
by letter.  Equate statements which define registers should, however, be
encountered in the source code before the assembler scans any instructions
using the register name, because register instructions vary in length
according to the registers being used.  Designating the symbolic name of a
register after the assembler has incremented the location counter past a
statement with the name in an operand can result in an erroneous value in the
location counter during assembly pass 1; all symbols stored in the symbol
table thereafter can be in error.  The assembler detects and reports this
error with an Invalid register value message.  The error can be avoided by
placing equate statements which rename registers at the beginning of a source
file.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  ʋ = question mark

## Linking Directives: DEF and REF

The define-symbol (DEF) instruction causes the assembler to include a symbol
and its value in the tagged-object file within a table used for resolving
symbolic references from other files.  The DEF instruction declares the value
of a symbol in the operand to be the value substituted while linking with
other files in which the symbol occurs undefined (provided the symbol is in
the operand of a REF instruction within the other files: see below).  The
symbol is "public" and available for use by other files being linked for
memory execution with the current file.

The external-reference (REF) instruction directs the assembler to put a symbol
in the operand into a list of tagged-object code symbols with values to be
determined from other files during the linking process.  Upon linking, symbols
tagged as REF symbols in one file are searched for among symbols tagged with
DEF in other files being linked: values in DEF tables corresponding with
entries in a program's REF list are returned to the program for use in
producing executable object code.

## End-of-Assembly Instruction: END

The END instruction causes any succeeding lines in a source file to be ignored
during both passes of the assembly.  The END directive is optional at the
physical end of a file being assembled.

## Multiple Source File Directive: COPY

The COPY instruction directs the assembler to include a source file other than
the one specified at the outset of assembly.  This instruction does not
actually copy source code between mass-storage files.  It does, however, at
the point of inclusion in the source file originally specified for assembly,
copy the file named in the operand into memory for continuation of assembly
during both passes.  When the copied file is completely scanned during either
pass of assembly, the assembler resumes its scan of the primary file specified
in response to the assembler Source file prompt with the instruction following
the COPY directive.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  ⌴ = percent sign;  ⌴ = at sign:  ⌴ = question mark

Files included in assembly by a COPY instruction can not contain COPY
instructions: only one level of source-file nesting is permitted.

If the filename in the COPY directive is preceded by the letter "P" and a
comma, the assembler issues a prompt for insertion of media containing the
file to be copied during all passes of the assembly.  For example, the
directive COPY P,2.COPYFILE produces an Insert: 2.COPYFILE message in the
display:

The COPY instruction also requires that both the original file and the file
being copied into it be open simultaneously.  Therefore, two mass-storage
devices such as Wafertape™ devices, which permit only one file to be open
at a time, must be used.  A third device is required if In memory assemble
(Y/N)⌴ has been answered with [N].

Listing Directives: IDT, TITL, PAGE, LIST, and UNL
If a device number or filename is entered in response to the List file prompt
at the beginning of assembly, an assembly listing is printed during the second
assembly pass.  This listing, an example of which is shown in figure 5-1,
consists primarily of the object code generated during assembly against the
left margin of the paper in correspondence with the source-code line from
which it is generated to its right.  (Note that the listing has been edited to
align the elements of each line into columns so that they can be labeled; the
actual listing has fewer spaces between elements of each line.)


    Figure 5-1.  Listing of an Assembled Program


Following assembly, the assembler can be directed to perform a third pass to
produce a cross-reference table, an example of which is shown in figure 5-2.
The cross-reference table correlates the name of each symbol used in the
assembled program, the value assigned to the symbol during assembly, the
number of the source-code line on which the value is assigned, and the numbers
of lines on which the symbol is referenced (in operands).

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  △ = percent sign;  c = at sign;  u = question mark

The page-eject instruction PAGE causes an ASCII form-feed character (>0C or
CTL-L) to be sent to the printer.  An ASCII form-feed characters can also be
sent automatically after a preset number of lines are printed on a page, as
specified in a LIST directive (see below).


Directives to List or Not List Lines

An entire program need not be printed in a listing.  Selected lines can be
listed by preceding them with a LIST instruction.  Selected lines can be
suppressed by preceding them with an unlist, UNL, instruction.  Lines
containing either the LIST or the UNL instruction are not printed in the
listing.


The LIST instruction can, in addition, be used with a numeric operand to set
the number of lines to be printed on a page before a page is automatically
ejected.  The instruction LIST 45, for example, causes 45 lines of text
(including the heading and blank line following it) to be printed before a new
page is begun.  A LIST instruction with the number 0 in the operand, however,
has a special purpose: LIST 0 suppresses the heading and all form feeds.  A
LIST 0 directive is in effect when listing begins.


A LIST directive without an operand causes printing to resume with the number
of lines on each page set at the same value as in the most recent LIST
directive with an operand.


Option Directive: OPTION

The OPTION directive provides for automatic deletion of listing lines which
consist of only object code produced by BYTE, DATA, and TEXT instructions.  It
also  provides for printing a cross-reference table of assembly variables.
Like other directives, OPTION is placed in the op code field of an
instruction, and letters representing selected options are placed in the
operand field.  Multiple selection of options in the operand field requires
that commas separate each letter, as in the following example.


    OPTION T,R

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign; Δ = percent sign; σ = at sign; υ = question mark

Figure 5-2. A Typical Cross-Reference Table

Both listings and cross-reference tables are controlled by assembler
directives. Some directives control listing format, including page headings,
title lines, and page length. Other listing-control directives control
whether selected parts of a listing are to be suppressed and whether a
cross-reference table is produced.

## Listing Format Directives

The file-identifier (name) instruction IDT specifies in the operand the name
of the file being assembled. This name appears, along with a page title and
page number, in a heading at the top of each page. The IDT directive also
causes the name of the program to be included in the assembler output file,
thus providing the linker with the program name. The name is also stored in
the object file for use by the linker in reporting on files successfully
linked. It can contain as many as eight characters and must be surrounded by
single quotation marks. If no IDT instruction is present, the default name
"NO$IDT" is printed in the heading.

The page-title instruction TITL causes additional text to appear in the
heading line beside the program identifier. The content of the line, typed
into the operand of the instruction and surrounded by single quotation marks,
can contain up to 40 characters. For a page title to appear on the first page
of the listing, the TITL instruction for it must precede any line which is to
be printed in an assembly listing. After a page title is specified, it
appears at the top of every page following the TITL instruction until it is
subsequently changed to the text in a subsequent TITL instruction. The line
containing the TITL directive is not printed in the listing. If no TITL
instruction is present to specify the title to appear in the heading, the
default title "CC-40 THS7000 Assembler Version 1.0" is printed.

| SOURCE PROGRAM LINE NUMBER | LOCATION OF OBJECT CODE | MACHINE OBJECT CODE IN HEX | LABEL | OP CODE | OPERAND(S) | COMMENTS |
|---|---|---|---|---|---|---|
| 0001 | | 0050 | COUNTER | EQU | >50 | COUNTER MEMORY LOCATION |
| 0002 | 0000 | | BEEP | | | |
| 0003 | 0000 | 88 | | MOVD | $>400,COUNTER | SET COUNTER TO >400 |
| | 0001 | 0400 | | | | |
| | 0003 | 50 | | | | |
| 0004 | 0004 | | BEEP2 | | | |
| 0005 | 0004 | A2 | | MOVP | $1,P21 | TURN ON THE BEEPER |
| | 0005 | 01 | | | | |
| | 0006 | 15 | | | | |
| 0006 | 0007 | 52 | | MOV | $29,B | |
| | 0008 | 10 | | | | |
| 0007 | 0009 | | ONLOOP | | | |
| 0008 | 0009 | CA | | DJNZ | B,ONLOOP | DELAY FOR ON PERIOD |
| | 000A | FE | | | | |
| 0009 | 000B | A2 | | MOVP | $0,P21 | TURN BEEPER OFF |
| | 000C | 00 | | | | |
| | 000D | 15 | | | | |
| 0010 | 000E | 52 | | MOV | $32,B | DELAY COUNT FOR OFF |
| | 000F | 20 | | | | |
| 0011 | 0010 | | OFFLOP | | | |
| 0012 | 0010 | CA | | DJNZ | B,OFFLOP | DELAY FOR OFF PERIOD |
| | 0011 | FE | | | | |
| 0013 | 0012 | 08 | | DECD | COUNTER | COUNT OF CYCLES DONE |
| | 0013 | 50 | | | | |
| 0014 | 0014 | E3 | | JC | BEEP2 | LOOP BACK UNTIL ALL DONE |
| | 0015 | EE | | | | |
| 0015 | 0016 | 0A | | RETS | | RETURN TO CALLING PROGRAM |
| 0016 | 0017 | 50 | | | | |
| | 0018 | 45 | | | | |
| | 0019 | 45 | | | | |
| | 001A | 42 | | RTEXT | 'BEEP' | PROGRAM NAME IN REVERSE CAPITALS |
| 0017 | 001B | 04 | | BYTE | 4 | NAME LENGTH |
| 0018 | | 001B' | SNAME | EQU | $-1 | |
| 0019 | 001C | 0000' | | DATA | BEEP | PROGRAM ENTRY POINT |
| 0020 | 001E | 0000 | | DATA | 0000 | NEXT SUBPROGRAM OFFSET |
| 0021 | 0020 | FFFC | | DATA | SNAME-$+1 | OFFSET TO NAME |
| 0022 | 0022 | 00 | | BYTE | 0,>44 | |
| | 0023 | 44 | | | | |

NO Errors   NO Warnings

Figure 5-1.  Listing of the Assembled Programs

```
A24                CC-40 TMS7000 Assembler Version 1.0              PAGE 0001

0002                    IDT   'A24'
0003            *
0004            * This is an example of a cross refrence map with copy files
0005            *
0006                    OPTION X              Generate cross reference map
0007       0001  X1      EQU   1
0008       0002  X2      EQU   2
0009            COPY   1,A24A
A0001          IDT   'A24A'
A0002
A0003    AA11  A1      EQU   >AA11
A0004    AA22  A2      EQU   >AA22
A0005 0000 AA11    DATA  A1,B1
      0002 BB11
0010            COPY   1,A24B
B0001          IDT   'A24B'
B0002
B0003    BB11  B1      EQU   >BB11
B0004    BB22  B2      EQU   >BB22
B0005 0004 AA22    DATA  A2,B2
      0006 BB22
0011 0008   62   MOV   X1,A
0012 0009 AA11   DATA  A1,B1
      000B BB11
 NO Errors   NO Warnings


A24                CC-40 TMS7000 Assembler Version 1.0              PAGE 0002

LABEL       VALUE   DEFN    REFERENCES

A1          AA11    A0003   A0005   0012
A2          AA22    A0004   B0005
B1          BB11    B0003   A0005   0012
B2          BB22    B0004   B0005
X1          0001    0007    0011
X2          0002    0008
```

Figure 5-2.   Typical Cross-Reference Table

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed

alpha = pound sign; Δ = percent sign; σ = at sign; u = question mark

---

The BYTE, DATA, TEXT, and RTEXT pseudo ops sometimes result in the printing of
many unwanted lines of object code. Each byte of object code produced by
BYTE, TEXT, and RTEXT (each word of object code produced by DATA) is listed on
a separate line beginning with the line on which the instruction occurs. A
30-character string in a TEXT statement therefore produces 30 lines of
listing. To eliminate such long segments of listing, four directives are
used. The OPTION B, OPTION D, and OPTION T directives cause only the initial
line of object code generated respectively by BYTE, DATA, and TEXT (or RTEXT)
statements to be listed. Listing of all object-code lines following the first
is suppressed. The OPTION F option cancels the effect of any and all OPTION
B, OPTION D, or OPTION T directives issued in the source code preceding it.

An X or an R option in the operand field of an OPTION directive causes the
assembler to perform a third assembly pass to generate a cross-reference table.

Both X and R produce identical tables if no file is being copied (included)
during assembly. If one or more files are copied, however, the two directives
produce different tables. The X instruction causes all symbols defined in
both the primary file and copied files to be listed. The R instruction also
causes all primary file symbols to be included in the table, but it reduces
the symbols included from the copied files to those which are referenced.

When copied files are assembled and cross-referenced, line-numbers in
cross-reference listings for symbols within them are keyed with alphabetic
characters. The letter "A" precedes line numbers in the first copied file
encountered during assembly, the letter "B" precedes numbers in the second
copied file, and so on. The cross-reference table in figure 5-2, above,
shows an assembly which includes two copied files.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  c = at sign:  υ = question mark

ASSEMBLY ERRORS AND ERROR MESSAGES

During passes 1 and 2, the assembler screens the source file for errors in
instruction entry.  If a listing is being printed during pass 2, the assembler
prints a message identifying any error on an unnumbered listing line
immediately following the instruction line on which the error occurs.  If no
listing is being printed, the assembler outputs the assembly-listing of the
line containing the error to the display.  Pressing any key brings any
subsequent lines of object code produced by the source line or the error
message into the display.

Assembly error messages do not necessarily indicate that the object code
produced by corresponding instructions is incorrect.  Error messages do mean
that the object code can be in error and should be examined.  Likewise, the
absence of assembly errors does not indicate that "error-free" programs can be
executed as expected.  The absence of assembly errors indicates the
acceptability of source-code instructions to the assembler on an
instruction-by-instruction basis; the logical flow of the program remains
unchecked until the program is executed and debugged.  (See chapter 8, "The
DEBUG Monitor.").

The three tables shown below list error messages and explain the errors which
correspond to each message.  The first table lists warning messages.  The
object code produced by a warning can be usable, but it should be inspected
carefully.  The second table lists errors classified as "severe errors."
Severe errors indicate a very high probability that incorrect object code has
been produced.  The third table lists errors classified as "fatal errors."
Fatal errors cause immediate and unrecoverable termination of assembly.
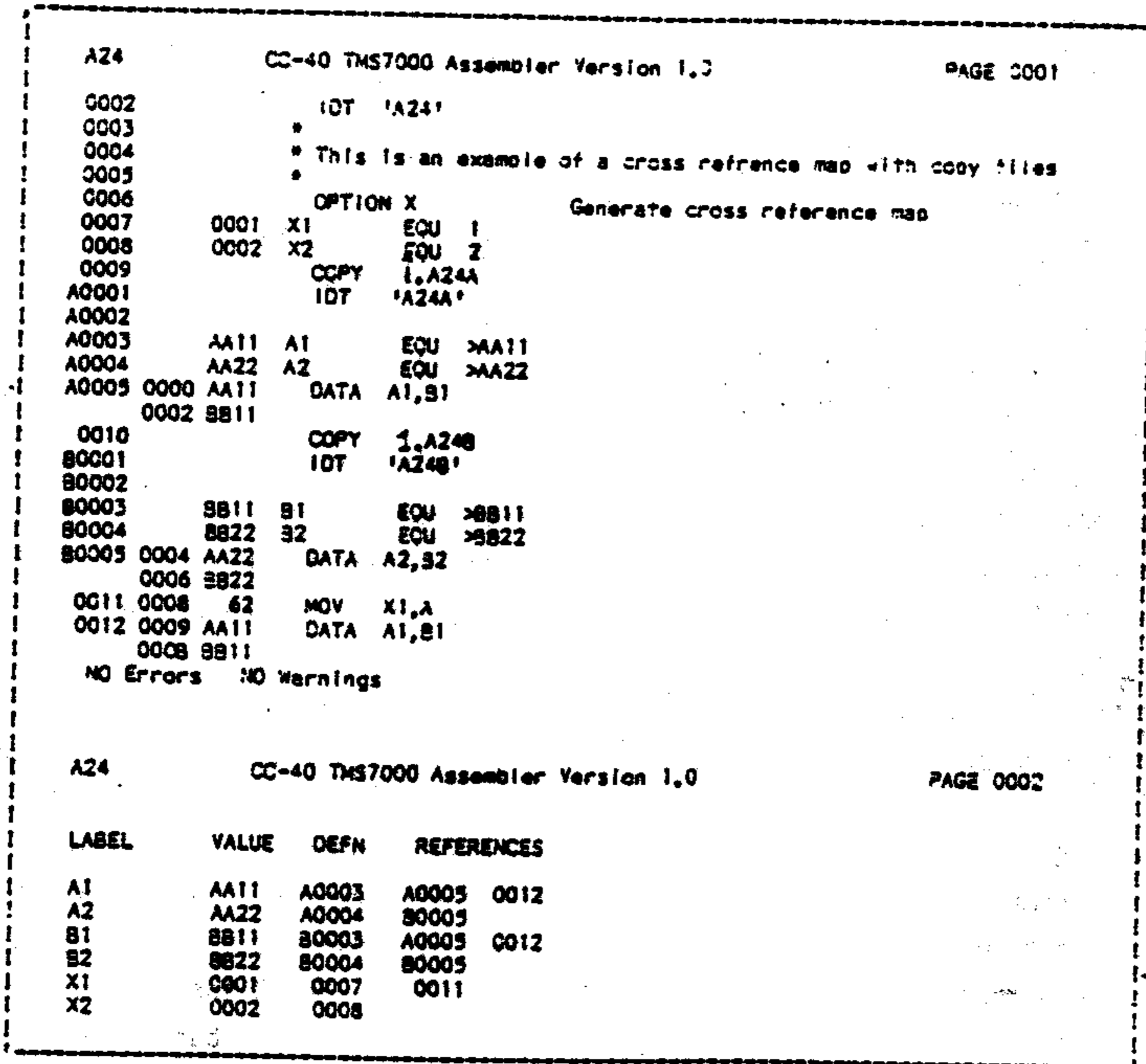
underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed

alpha = pound sign; Δ = percent sign; ? = at sign; u = question mark

## WARNING MESSAGES

| MESSAGE | EXPLANATION |
| --- | --- |
| Value truncated | A numeric value is too great. For example >1FFFF has been evaluated to a word expression as >FFFF |
| Symbol truncated | A symbol string contains more than eight characters (only the first eight are significant) |
| Trailing operands | Too many expressions in the operand field: extra expressions ignored |

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign; & = percent sign; c = at sign; u = question mark

## SEVERE ERROR MESSAGES

| MESSAGE | EXPLANATION |
|---|---|
| Invalid op code | The command field does not contain a valid instruction mnemonic, pseudo operation, or directive |
| Undefined symbol | An expression in an operand field has not been defined by a label or a REF operand, and it cannot be evaluated |
| Multiply defined | A symbol is defined more than once symbol |
| Invalid symbol | A symbol contains characters other than "A"-"Z", "a"-"z", "0"-"9", ">", and "$" |
| Invalid constant | A string constant contains an invalid delimiter or a numeric constant contains an invalid character |
| Invalid expression | Expression contains illegal operations or invalid symbols |
| Missing operand | An operand required with this op code is not present |
| Invalid operands | Operand(s) present are not valid for op code |
| Invalid use of symbol | A symbol was REFed and DEFed, or a REFed symbol is used as a label |

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

| | |
|---|---|
| Reserved symbol | "S","A","B","R0"-"R127","R>0"-"R>7F", "P0"-"P255", "P>0"-"P>FF" or "ST", all predefined symbols, are being redefined |
| Invalid register value | A register value exceeds 255, is not absolute, or contains an undefined value (external or forward referenced symbol) |
| Divide by 0 | Expression contains division by 0 |
| Too many nested parentheses | Expression has too many nested parentheses parentheses (normally five levels): simplify the expression |
| Missing ")" | Expression lacks closing parenthesis of pair |
| Syntax error | Invalid syntax in an operand |
| Displacement too big | Offset generated by the jump instruction out of -128 to +127 value range |
| Mixed module | The source contains both absolute and relocatable code (AORG and RORG) |

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed

alpha = pound sign:   % = percent sign:   @ = at sign:   ? = question mark

## FATAL ERROR MESSAGES

| MESSAGE | EXPLANATION |
|---|---|
| Memory full | The source, object, and/or copy files and the symbol table do not fit simultaneously in memory (reducing record sizes can remedy the difficulty) |
| PASS1/PASS2 conflict | A symbol is defined only in pass 2 or it has a value different in pass 2 from in pass 1, usually because of undetected file error or a different file being used during the second pass.  This is also a severe error if a symbol is defined recursively, as in X EQU X+1. |
| Nested copy files | A copied file contains a COPY directive |

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  £ = percent sign;  c = at sign;  y = question mark

# CHAPTER 6
# THE LINKER


This chapter describes the way the linker is used to convert tagged-object
code into machine language object files and to  link multiple assembler-output
files together into a single machine language program.

Assembler-output files must be processed by the linker before the object
program can be executed.  Assembler-output files are in tagged-object format
(see chapter 5 and appendix E of the CC-40 Editor/Assembler Reference
Manual).  These files must be converted into TMS7000-family machine-language
commands.

In addition to converting file format, the linker resolves (assigns values to)
references which have been produced by assembler REF and DEF directives.  It
produces either (1) relocatable-code programs or subprograms which can be
loaded and run under BASIC or (2) relocatable- or absolute-code programs which
can be loaded and run in CC-40 system or cartridge RAM.  Programs output by
the linker contain either absolute code or relocatable machine code (the
assembler does not produce an output file from a mixture of both).

Like assembly, linking requires two passes.  The first pass determines the
length of the object file and builds a symbol table for references among
files.  The second pass produces a memory image of the object code (and a
relocation table if the object code is relocatable).


## RUNNING THE LINKER
After the ALDS is entered from BASIC with a [RUN "ALDS"] command, the ALDS
program menu appears in the console display as shown below.

     E)dit A)ssemble L)ink B)asic

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;   Δ = percent sign;   𝒿 = at sign:   ʉ = question mark

Pressing [L] when the menu appears causes the linker to begin execution.

Linking is controlled by commands stored in CC-40 memory.  The commands can be
entered directly on the keyboard at the time of linking, or they can be read
in from mass-storage after they have been created and saved by the ALDS editor.

Upon entry, a linker prompt asks if there is a control file on a mass-storage
device which is to control the linking process.  The prompt Is there a control
file (Y/N)ʉ appears in the display.  The control file contains linking
commands entered through the ALDS editor.

·If [N] is pressed, the display clears and the linker prompts with Enter line
for keyboard entry of commands to control linking.  Each command is entered on
a single line.  The last line entered is an E command which signals the end of
the commands list.  As each command is entered, the linker checks it for
errors and displays any appropriate error message.

If [Y] is pressed in response to the prompt, the linker then prompts for the
control-filename.  When the control-filename is entered, the linker opens the
file and scans it for any incorrect linker commands.  The scanning continues
until an E command code is found or until the end of the control file is
reached.  If the end of the file is reached before an E command is found, a
warning message is displayed and an E command code is appended to the file.
Any other errors found  are printed on a "link map," a comprehensive listing
which records the result of the linking process.

After the linker receives commands from the keyboard or reads from the control
file, it prompts for an output filename with the prompt Linked output file.
If no name is given, no object code is output.  The code is nevertheless
generated internally so that errors can be determined.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  & = percent sign;  @ = at sign;  u = question mark

---

Next, the linker prompts for the HEX-BUS[TM] device which is to print the link
map. A listing can be printed regardless of whether an object-code file is
written. If no list filename is entered, no link map is printed.

## LINKING COMMANDS

Five commands control linker operation. These commands identify the
assembler-output files to be linked, the type of machine code to be produced
(absolute or relocatable), and the format of the linker-output file. Each
command consists of a single-character code followed by any appropriate
argument or parameter.

Spaces are allowed—but not required—between a command code and an argument.
Only one command can appear in each line. Lines preceded by an asterisk (*)
permit comments within a link-control file. Comments appear in the link map.

### Input-File Commands

An input file is included in the linking process if it is named in an include
command code (I or J). The argument for either include command is the name of
the file to be included. The J command code is identical to I, except that J
prompts the user to mount a mass-storage medium (Wafertape[TM] medium, for
example) with the file on it. From a J code, a message such as Insert:
2.PROG2 appears in the console display.

As many as 255 input files can be processed in a single link operation.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  7 = at sign:  u = question mark

## Absolute-Output Command

The absolute-address command A results in the output code which is absolutely
addressed beginning with the value in the argument. If no A command code is
found by the linker (and an AORG instruction from the assembler is not found
before any code-producing assembler-output command is found, the output code
defaults to relocatable code.  If the linker is processing both relocatable
and absolute files and the first file specified in an I or J command is a
relocatable file, an A command must precede the command specifying the file.
The linker can process combinations of absolute and relocatable input files
into an absolute output file, but it cannot process such combinations into a
relocatable output file.

As in assembler AORG instructions, linker A commands must have arguments in
which subsequently specified absolute addresses are higher than any address
previously assigned in the linking process.  The file currently being linked
cannot produce code with memory addresses lower than or the same as addresses
used by code from a previous file.

## Output-File Format Command

The multiple-record output command code M sets the linker to produce an output
file containing more than one record of code.  The argument for the M command
is the record length.  The M command must precede the specification of any
assembler-output filenames (see the I command, above).  If an M command is not
present, the linked file is accumulated in memory for subsequent writing to
mass-storage at the end of linking, either as a single-record relocatable-code
file or as a two-record absolute-code file.

Only one M command can be issued in a single linking operation.  When
multiple-record output files are to be produced and mass-storage devices such
as Wafertape™ units (which permit only one open file to be open at one
time) are being used, the output device must be separate from the input device.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  υ = question mark

## The END Command

The E command code signifies the end of the commands issued to the linker.

## LINKER COMMAND TEXT AND LISTING

The following figure shows a series of commands provided to the linker by a
control file.  Each valid command is represented, and one invalid command is
included.

```
A         >9000
* START THE CODE AT THE BEGINNING OF A 16K CARTRIDGE
M         255
* NOTE THAT THE FOLLOWING ARE FILE (NOT PROGRAM) NAMES
I         1.CARTHEAD
I         1.PROG1
J         1.PROG2
* NOTE THAT THE FOLLOWING COMMAND IS INVALID
Z
E
```

When a link map listing is produced, the first item in the map is a list of
the commands.  Errors in the control file are documented in the map, as shown
in the figure below.  Note that the commands in this figure must have been
issued in a control file, because the bad command code Z would not have been
accepted in keyboard entry of command text by the linker.  Also note that if
Wafertape™ devices are being used, a device other than device 1 must have
been specified in response to the Linked output file prompt because of the M
command.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  c = at sign;  u = question mark

In a link map, the M command is always moved to the top of the command
listing, and the value of the argument it contains is always expressed in a
decimal number.


```
M        255
A        >9000
* START THE CODE AT THE BEGINNING OF A 16K CARTRIDGE
* NOTE THAT THE FOLLOWING ARE FILE (NOT PROGRAM) NAMES
I        1.CARTHEAD
I        1.PROG1
J        1.PROG2
* NOTE THAT THE FOLLOWING COMMAND IS INVALID
Bad command code: Z
E
```


## LINK MAP FORMAT

In a link map, the beginning addresses and lengths of each module
(assembler-output file) code are listed in hexadecimal values beside the names
assigned in assembler IDT instructions for each file. For identification in
error messages and the symbol (definition) table in the link map, the linker
numbers modules with decimal numbers in the order in which they are linked.
The following figure shows a typical link map with one error identified. A
public symbol called "SCAN" has multiple definitions: it is defined or "DEFed"
in two modules, once in the module with the IDT name NOSIDT and a second time
in the module with the IDT name 01234567.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
#lona = pound sign; Δ = percent sign; ɛ = at sign; υ = question mark

CC-40 TMS7000 LINKER   Version 1.0

## CONTROL FILE

```
M    64
* PUT OUT 64 BYTE RECORDS
I    1.FILE1
Bad command code:  TYYYY
I    1.FILE2
I    1.FILE3
E
```

## MODULE MAP

| NUMBER | MODULE   | ADDRESS | LENGTH |
|--------|----------|---------|--------|
| 1      | NO$IDT   | 0000    | 0034   |
| 2      | 01234567 | 0034    | 0072   |
| 3      | XY       | 00A6    | 018E   |
|        |          | 0233    | 0234   |

*** Warning SCAN in ɛ    2 already defined in ɛ    1

## DEFINITIONS

| NAME | VALUE | NO | NAME | VALUE | NO | NAME | VALUE | NO |
|------|-------|----|------|-------|----|------|-------|----|
| ASTART | 0000' | 1 | BLOOP | 0049' | 2 | DLOOP | 0023' | 1 |
| FLOOD | 0043 | 2 | GO | 0093 | 2 | HGG | 0045' | 2 |
| *INPUT | 0190' | 3 | METER | 0069' | 2 | *NEXT | 00A6' | 3 |
| *ONLY | 0234' | 3 | SCAN | 0014' | 1 | *SCAN | 0053' | 2 |
| *ZREF01 | 0119' | 3 | | | | | | |

****    2 : UNRESOLVED  REFERENCES

| NAME | NO | NAME | NO | NAME | NO | NAME | NO |
|------|----|------|----|------|----|------|----|
| Z2 | 2' | ENTRYPT | 3 | | | | |

LINK COMPLETE

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign; ∆ = percent sign; ⌀ = at sign; u = question mark

In the module map portion of the link map, relocatable values are marked with
single quote marks following the assigned value. In the example, all symbols
except "FLOOD" and "GO" are address-related in the three files which have been
assembled to produce relocatable code. Symbols preceded by asterisks are
"DEFed" in the numbered files, but they are not referenced or "REFed" in other
files.

When symbols are defined more than once, the value assigned to the symbol is
the value associated with the first DEF instruction found. Occurrences of the
symbol after the first DEF instruction appear in the listing, but they are
marked with asterisks because they are not REFed.

Following the symbol table, unresolved references are listed. There are no
DEFed symbols to correspond to the listed REFed symbols.

LINKER INPUT AND OUTPUT FORMATS
Assembler output files—input files for the linker—contain tagged-object
code, are described in appendix E of the CC-40 Editor/Assembler Reference
Manual.

Output from the linker is a file which is a memory image of the program to be
executed. If the file contains relocatable object code, memory-image code is
followed by a relocation table containing offsets to words of code which must
be relocated prior to program execution.

HEADERS, MEMORY-IMAGES, AND RELOCATION-TABLE RECORDS
Unless an M command is in effect, relocatable code is output in a single data
record for loading, relocating, and running under BASIC. The single record
has no separate header record, as shown below.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  & = percent sign;  o = at sign:  v = question mark

```
Beginning of File                        End of File

        ,--------------------------------------,
        :                      :               :
        :   Relocatable code   :  Relocation   :
        :                      :               :
        :                      :    Table      :
        :                      :               :
        '--------------------------------------'

        One record
```

Absolute code which is output with no M command in effect is saved in a file
which contains two records.  As shown below, the first record contains a
header and the second contains a memory image of the absolute code.  The
memory image of the absolute code is filled with NOP commands (>00) between
segments of code set with AORG instructions not to be contiguous.  When a
relocatable code module follows absolute code, the relocatable module becomes
absolute and machine code from the module immediately follows the last byte of
the previous module.

```
Beginning of File                        End of File

   ,---------,     ,------------------------------------,
   :         :     :                                    :
   !Header   !     !Absolute code with addresses for    !
   :         :     :                                    :
   !Record   !     !execution already calculated        !
   :         :     :                                    :
   '---------'     '------------------------------------'

   Record 0        Record 1
```

With an M command in effect, absolute code is output as a memory image
multiple-record file with a header record, as shown below.  The length of the
records is specified in the M (although the last record can be less than the
record length specified if fewer bytes remain to be written).

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign; ▲ = percent sign; σ = at sign; u = question mark

Beginning of File                        End of File

```
,--------, ,----------, ,----------, ,--------,
!header !  !record   !  !record   !  !record  !
!record !  !with     !  !with     !  !with    !
!       !  !lowest   !  !higher   !  !highest !
!       !  !address  !  !address  !  !address !
'--------' '----------' '----------' '--------'
```

  Record 0   Record 1    Record 2    Record 3


With an M command in effect, relocatable code is also output as a memory-image
following a header. Additionally, a relocation table is written as the last
record of the file, as shown below. The relocation table is saved as a single
record, regardless of the length specified in the M command.


Beginning of File                              End of File

```
,--------, ,----------, ,----------, ,--------, ,------------,
!header !  !record   !  !record   !  !record  !  !relocation  !
!record !  !with     !  !with     !  !with    !  ! table      !
!       !  !lowest   !  !higher   !  !highest !  !            !
!       !  !address  !  !address  !  !address !  !            !
'--------' '----------' '----------' '--------' '------------'
```

Record 0   Record 1    Record 2    Record 3  Record 4


HEADER RECORDS

An 11-byte header record contains the six fields of information used in
subsequent loading of linked files.


   1.  A file-type word, set to >0F05 to indicate a linker-output file


   2.  A flag byte, indicating whether the file contains relocatable or
       absolute code and a single- or multiple-record memory image (and
       relocation table)

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  ∆ = percent sign;  ¢ = at sign;  u = question mark

    3.  A word specifying the maximum memory space required for writing the
       relocation table into memory

    4.  A word specifying record length for multiple-record files

    5.  A word specifying memory-image code length

    6.  A word specifying either absolute load address or minimum
       relocation-table length for multiple-record files

The function of each of the header-record fields is as follows.


## File-Type Word

The file-type word, stored in memory at the highest address of the header
block, has the value >0F05.  This word uniquely identifies linker output files
which contain absolute-code or multiple records of relocatable code so that
they cannot be erroneously read by BASIC and loader programs for which they
are not designed.


## Flag Byte

The flag byte indicates whether a file is written in multiple records and
whether the file contains relocatable or absolute code.  The individual flags
in the byte are set in accordance with the hexadecimal values shown in the
following table.


### FLAG-BYTE VALUES

|  | TYPE OF CODE OUTPUT | |
| RECORD OUTPUT | Relocatable | Absolute |
|---|---|---|
| Single | >04 | >00 |
| Multiple | >84 | >80 |

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

## Maximum Relocation Table Length

The maximum length (in bytes) of a relocation table is two more than twice the number of relocatable words and external references (REF's) in the program.

## Block Size

The block size provides a loader with information required to load multiple-record images into memory (from high to low address) for the correct load address.

## Length of Memory Image

The length of the memory image (without relocation table) permits a loader to verify that a program can fit into available memory.

## Absolute Load Address or Minimum Relocation Table Length

This dual-purpose header word contains different information about absolute images and relocatable images.

For absolute images, it provides the load address for the start of absolute code.

For relocatable images, this word and the maximum relocation table length permits a loader to verify that sufficient memory is present for loading the file.  The minimum relocation-table length is calculated to be two more than the number of relocatable bytes in the program.

## ERROR MESSAGES

The following tables lists error messages resulting from control-file errors and errors occurring during linking.

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
alpha = pound sign;  ∆ = percent sign;  ∂ = at sign;  u = question mark

## CONTROL-FILE ERRORS

| | |
|---|---|
| Multiple M code | Second M command code found in control file. Second one is ignored. |
| M code after include | M code found after first I or J command. Second M code has been ignored. |
| Bad value | Incorrect or missing number found in control file. Command is ignored. |
| Bad filename | Illegal filename found in control file. Command has been ignored. |
| Bad command code X | "X" is not a legal linker command |
| Missing E command code | No E command code found before end of linker control file. Appears in display. |
| Too many files to be linked | More than 255 I and J control codes are used in a single control file. |
| Bad file type | Control file is of the wrong type (not a text file without special characters). |

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  ʯ = question mark

___

### ERROR MESSAGES OCCURRING DURING LINKING

Code wraps around            Code generated for beyond address >FFFF.
address space                A warning only for absolute code, but also causes
                             fatal memory-full error for relocatable code.


Memory full                  More RAM needed than available


Too many errors              Too many errors (multiply defined symbols) and
                             extra module (IDT-specified) names


Missing EOR tag              End-of-record tag in assembler object code
in α N                       missing.


Missing module               Module name tag not the first item found
name in α N                  in tagged-object file.


Bad program type             File of incorrect type specified in
type in α N                  control file. Not an assembler-output
                             tagged-object file with >0F04 in the header.


Missing load                 No load address tag in object code found
address in α N               before tag generating memory image code found.


Mixed code in                Both absolute code and relocatable code in
module in α N                a module


Pass1/Pass2 data             Data in the first pass and the second pass
conflict in α N              is different (possibly two files of same name or
                             mass-storage read error).


Code overlaps existing       Load address tag less than the current
code in α N                  address being linked.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  z = percent sign;  c = at sign;  u = question mark

Abs address <                Address in A command is less than the
current address             current absolute address of the code being
in a N                       linked.  Number is of last module linked.

Extra module                 A second module name tag found in a
name in a N                  single tagged-object file. The second name is
                             ignored.

*** Warning XXXXX            The public symbol "XXXXX" has been "DEFed"
in a N2 already             in two files and is a multiply defined
defined in a N1             symbol.

Illegal tag                  Illegal assembler tag has been
XX XX XX XX XX              encountered. Bad tag and the 4 following
in a N                       bytes are displayed or printed.

Checksum error               File checksum calculated by the linker
in a N                       does not agree with the checksum generated by the
                             assembler.

Absolute in                  Absolute code found after relocatable code
relocatable link            (possibly AORG after RORG in assembly).
in a N

underlining in text (except heads) indicates a typestyle other than normal text: single brackets indicates information to be typed; double brackets indicates key to be pressed
alpha = pound sign: / = percent sign; c = at sign; y = question mark

CHAPTER 7

ALDS UTILITY PROGRAMS

This chapter describes the use of ALDS programs which provide support for the development of programs written in assembly language (and BASIC as well). These utility programs work with the files which are input to and output from the editor, assembler, and linker during assembly-language program development.

The Editor/Assembler cartridge contains ALDS utility programs which are used to print or display files, to convert files to various formats used by the CC-40 and other computers, and to transfer files. The utility programs include the following.

!o! A hexadecimal file-dump utility to permit output of the contents of any ALDS file.

!o! A cartridge loader and an absolute-code loader which load—for subsequent execution—linked object files into CC-40 cartridge RAM or internal RAM.

!o! A cartridge-save utility which writes the contents of a RAM cartridge into a mass-storage file.

!o! Several file-conversion and -transfer utilities: programs which convert between ALDS-format files and files from external computers, from the TI-990 assembler or linker, and CC-40 BASIC; and programs which provide for RS232 data transmission to transfer these files.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  ﹩ = percent sign;  ﻭ = at sign;  ﹖ = question mark

HEXADECIMAL DUMP UTILITY (HEXDUMP)
The hexadecimal dump utility (HEXDUMP) outputs each byte in every record of a
file as both a hexadecimal number and an ASCII character. Files which are
output by the editor, the assembler, the linker, and other utility programs
can be examined in detail.

Running the Hexdump Program
When [RUN "ALOS"] is entered in response to the BASIC cursor, a partial
display of the ALOS menu appears in the display as shown below.

   E)dit A)ssemble L)ink B)asicu

Pressing the [[SPACE BAR]] shifts the window to show the three remaining ALOS
commands, as shown below.

   H)ex dump D)ownload T)ransferu

Pressing [H] while either part of the menu is displayed causes the CC-40 to
run HEXDUMP. An input file message prompts for entry of the name of the file
to be dumped. After the source of the file is entered, an Outout file prompt
appears. After the destination is entered and the output device is opened,
the prompt Memory image file (Y/N)u appears.

A [Y] response to the memory-image prompt causes the dump to be output in the
order in which it appears in memory (from lower to higher addresses). An [N]
response causes the dump to be output in the order in which it is read into
memory from the mass-storage device (from higher to lower addresses).

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign:  A = percent sign;  c = at sign:  u = question mark

Following the [Y] or [N] response to the memory image prompt, the dump
begins.  It continues until [[BREAK]] is pressed or until the last byte of the
last record in the input file is printed or sent to an output file.  If
[[BREAK]] is being held down when an input/output operation occurs, HEXDUMP
operation is interrupted with the prompt BREAK, Continue (Y/N)u  Pressing [Y]
causes the dump to continue.  Pressing [N] causes the ALDS menu to appear once
more.

## Use of Hexdump

The Memory image file (Y/N)u option provides for dumping both text files and
machine-code files in the most readable order.

Each text area created by the ALDS, to make most powerful use of the
7000-family DECD (decrement double register) instruction, has text and tokens
stored from the top of the area downward in memory.  (Mass-storage of an area
has no effect on the order of bytes in text; bytes read back into memory are
in the same order in memory as when they were written to a mass-storage
device.)  The result is that a text file, if it is viewed in the order of
ascending memory locations, appears to be "backwards."  An answer of [N] to
the Memory image file (Y/N)u prompt causes the bytes to appear in "correct"
order so that text displayed in the ASCII portion of the display is "readable."

In memory-image sequence, bytes from a machine-language record are output in
the bottom-to-top order in which they are executed in memory.  HEXDUMP outputs
bytes stored at lower memory addresses first.

Each record output by HEXDUMP is preceded by a heading which lists the record
number.  The location of each byte in the record is referenced by column and
row numbers respectively at the top and left of the hexadecimal area of the
output.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign; Δ = percent sign; σ = at sign; μ = question mark

Sixteen bytes of a record appear in each line of HEXDUMP output. The bytes
are first output in two-digit hexadecimal values separated by spaces.
Following the hexadecimal representations of the bytes, 16 ASCII
representations of the bytes are output. When a byte cannot be represented by
a printable ASCII character, a dot (period) is printed in its place.
A HEXDUMP output of the first and last four lines in a text file which
contains the source code for the "BEEP" program appears in the following
figure.

```
,---------------------------------------------------------------------------,
: OPENED FILE:   1.SRC                                                       :
:                                                                           :
: RECORD      1                                        --*                   :
:                                                                           :
: NUMBER    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F                   :
:  0000 OF 00                                                      ..        :
:                                                                           (
: RECORD      2                                                              :
:                                                                           :
: NUMBER    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F                   :
:  0000    43 4F 55 4E 54 45 52 20 20 20 45 51 55 20 20 3E   COUNTER   EQU  > :
:  0010    35 44 20 20 20 20 20 20 20 20 20 20 20 20 43 4F   5D            CO :
:  0020    55 4E 54 45 52 20 4D 45 4D 4F 42 59 20 4C 4F 43   UNTER MEMORY LOC :
:          - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -   :
:  0330    46 53 45 54 20 54 4F 20 4E 41 4D 45 00 20 20 20   FSET TO NAME.    :
:  0340    20 20 20 20 20 20 20 42 59 54 45 20 30 2C 3E 34            BYTE 0,>4 :
:  0350    34 20 20 20 20 20 20 20 20 20 20 20 48 45 41 44 45   4        HEADE :
:  0360    52 20 49 4E 46 4F 52 4D 41 54 49 4F 4E 00           R INFORMATION. :
'---------------------------------------------------------------------------'
```

The >0F byte which begins record 1 indicates that the record is a header
record. The header contains the single byte (a >00 file-type flag) which
identifies the file as an uncrunched text file.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alone = pound sign; & = percent sign; = = at sign; p = question mark

Record 2 contains the text of the source code. Note the null byte (>00) which
separates each line within the record and the representation of the null byte
by a dot, because it is an unprintable character.

See chapter 5 for comparison of the dump with the assembler-output listing of
the source code for this file.

The following figure shows a complete HEXDUMP output of the tagged-object file
output from the assembler when 1.SRC is assembled.

```
,--------------------------------------------------------------------,
! OPENED FILE:   1.OBJ                                                :
!                                                                     :
! RECORD    1                                                         :
!                                                                     :
! NUMBER    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F             :
!  0000     OF 04 41 00 00 4E 4F 24 49 44 54 20 20 44 00 00   ..A..NO$IDT  D.. !
!  0010     43 88 49 04 00 49 5D A2 49 01 15 49 52 1D 49 CA   H.I..I].I..IR.I. !
!  0020     FE 49 A2 00 49 15 52 49 20 CA 49 FE DB 49 5D E3   .I..I.RI .I..I]. !
!  0030     49 EE 0A 49 50 45 49 45 42 48 04 4A 00 00 49 00   I..IPEIEBH.J..I. !
!  0040     00 49 FF FC 49 99 44 4C EA A8 4F                                   !
'--------------------------------------------------------------------'
```

Note that the tagged-object dump shows most of the code to be made up of
fields containing "I" (absolute data) tags followed by two-byte words
containing commands and data for the header. The initial >OF byte is a header
identifier, and the >04 byte following it indicates that the file contains
compressed tagged object code. The single-byte "0" tag identifies the end of
the file.

See appendix E of the CC-40 Editor/Assembler Reference Manual for a complete
description of the tagged-object code shown in this dump.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  ϵ = at sign;  ц = question mark

The following figure shows a dump of the linker output file as a single-
record relocatable code file for loading as a subprogram under BASIC's CALL
LOAD subprogram.

```
,--------------------------------------------------------------------,
: OPENED FILE:   1.BEEP     Memory image file                        :
:   '                                                                :
: RECORD 1                                                           :
:                                                                    :
: NUMBER     0. 1  2  3  4  5  6  7  8  9  A  B  C  D  E  F           :
:  0000     00 04 00 10 88 04 00 50 A2 01 15 52 1D CA FE A2   .......]...R....  :
:  0010     C0 15 52 20 CA FE 08 5D E3 EE 0A 50 45 45 42 04   ..R ...]...PEEB.  :
:  0020     C0 00 00 00 FF FC 00 44                            :
'--------------------------------------------------------------------'
```

The ASCII columns of the dump show with significance only the subprogram name
(stored "in reverse" by the RTEXT directive from the assembler).  The
hexadecimal columns show the most significant information: a four-byte
relocation table at the start of the record and the machine code of the
program.  The relocation table begins with a two-byte word specifying the
length of the table (including the length word itself) and contains a single
offset to the only relocatable word—the program entry point in the header—in
the code.

## Hexdump Error

A record output by HEXDUMP must reside completely in memory.  If not enough
memory is available for loading the record, a Memory full error message is
displayed.  The ALDS menu returns after any key is pressed.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  % = percent sign;  @ = at sign;  ? = question mark

OBJECT-FILE LOADERS
Two loader programs are present in ALDS ROM: an absolute-code loader called
ABSLOAD and a cartridge-RAM loader called CARTLOAD. Loading absolute programs
anywhere in system or cartridge RAM requires ABSLOAD.  Loading relocatable
programs into RAM cartridges so that they can be run under BASIC requires the
use of CARTLOAD.

Saving and Loading ABSLOAD and CARTLOAD
The CARTLOAD utility cannot be run while the Editor/Assembler cartridge is
being used, and the ABSLOAD utility cannot be used to load RAM cartridges
while the Editor/Assembler cartridge is in place.  The Editor/Assembler
cartridge overlaps much of the area of memory into which programs might be
loaded.  Therefore, loaders must be "downloaded" (saved from the cartridge) to
mass-storage media for later loading under BASIC when the Editor/Assembler
cartridge can be replaced by a RAM cartridge.

The D)ownload option in the ALDS menu provides the means for saving the
loaders (and also the MEMSAVE utility described below).  The loaders are saved
as single-record subprograms containing relocatable object code for loading by
BASIC with the CALL LOAD subprogram.

From the ALDS menu, [D] is pressed (as prompted by the second part of the ALDS
menu).  The downloader menu, shown below, then appears.

    M)emsave A)bsload C)artload

The loader to be saved to a mass-storage file is selected from the downloader
menu. The Output file prompt then requests the name of the file in which the
loader is to be stored.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed ·
alpha = pound sign;  Δ = percent sign;  ç = at sign:  μ = question mark

With the loader code in a mass-storage file, CC-40 cartridge RAM can replace
the ALDS ROM cartridge, and the loaders can be loaded into CC-40 internal RAM
by a BASIC CALL LOAD subprogram, as in the following examples.


     CALL LOAD("1.ABSLOAD")


   ,  or


     CALL LOAD("1.CARTLOAD")


## Absolute-Code Loader

Programs produced by the linker in absolute code are loaded into internal RAM
or cartridge RAM by the absolute loader.


After the absolute loader is copied from mass-storage by the CALL LOAD
subprogram, it is run from BASIC with the CALL command, exemplified by the
following.


     CALL ABSLOAD("1.PROG")


When the BASIC cursor returns (provided that there are no I/O errors reported
and that the BASIC environment has not been altered by the loaded program),
the absolute-code program is present in memory at the address set during
assembly or linking.  The program can then be executed from BASIC by a EXEC
subprogram such as the following, in which the entry address (the decimal
equivalent of >9000 is given as the argument for the command.


     CALL EXEC (36864)


The following table explains error messages which may appear in the display
when ABSLOAD is executed.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign; % = percent sign; c = at sign; u = question mark

## ABSLOAD ERROR MESSAGES

Illegal syntax        Bad syntax in the assembly language call

Can't do that        An absolute address of the program conflicts with the
                     memory space used by ABSLOAD.

Bad program type     The input file is not in absolute format

## Cartridge Loader

The cartridge loader loads relocatable assembly language programs and
subprograms into cartridge RAM to reside within >5000 through >CFFF.

The cartridge loader initializes CC-40 RAM cartridges and loads relocatable
code developed by assembly-language programs and subprograms into cartridge
RAM. The cartridge loader consists of two subprograms, CARTINIT and CARTLOAD.

## The CARTINIT Subprogram

The CARTINIT subprogram is called from BASIC with the name of the file to be
loaded inside of double quotes which are inside of parentheses, as in the
following.

    CALL CARTINIT("1.EXAMPLE")

The file must be in multiple-record, relocatable format.

The primary purpose of CARTINIT is to load and relocate a machine-code
cartridge header into cartridge RAM at >9000. This header enables BASIC to
access programs and subprograms in a cartridge by pointing to the first of the
programs the cartridge contains.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  ⅄ = percent sign;  ♂ = at sign;  u = question mark

The file loaded by CARTINIT can also contain one or more programs linked
together by offsets in their individual program headers.  In the cartridge
header, the pointer to the first program must be initialized with the
program-header address, and the offset in the header of the last program must
be initialized with >0000.  If no programs are included in the file, the
pointer in the header is initialized with >0000.

The following source program illustrates the relationship of cartridge and
program headers.  A complete explanation of headers is included in chapter 1
of the CC-40 Editor/Assembler Reference Manual.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  £ = percent sign;  c = at sign;  y = question mark

```
4000  BYTE     >A5,>5A     Tell BASIC that a cartridge is present
9002  BYTE     >81         16K cartridge; version 1
0003  BYTE     >1E         Cartridge access speed set to 3 microseconds
9004  DATA     SUB1        Pointer to header of first subprogram
9006  DATA     0           Reserved
9008  DATA     TOPCART     Pointer to next RAM location for loading
900A  DATA     >5000       Pointer to lower next RAM location for loading
```


* ASSEMBLY LANGUAGE PROGRAM (SUBROUTINE)


ENTRY
```
    MOV      A,B         A trivial program for illustration
    RETS                 Return to calling statement
```


* ASSEMBLY LANGUAGE PROGRAM HEADER


NAMLOW
```
    RTEXT    'SOMNAME'       Subprogram name
    BYTE     NAMHGH-NAMLOW   Length of the subprogram name
NAMHGH       EQU $-1
    DATA     ENTRY           Pointer to entry point
    DATA     0000            Offset to next program/subprogram header
    DATA     NAMHGH-$+1      Offset to name
    BYTE     0,>44           Program-type information
SUB1         EQU $-1
TOPCART      EQU $           First free byte in cartridge
```


The name of the program in each program header must be assembled from
upper-case (all-capitals) letters for BASIC to find it.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to oe typed; double brackets
indicates key to be pressed
alpha = oound sign;  Δ = percent sign;  σ = at sign;  ц = question mark

Cartridge RAM addressing is dependent on the type of cartridge being used and
the size of the memory chips used in it.  The illustration below shows the
address blocks corresponding to RAM chips currently available in cartridges.

```
>0000   ,-------------,           >0000   ,-------------,
        : 2nd 8K      :                   : 1st 8K      :
        : chip        :                   : chip        :
        : repeated    :                   : repeated    :
>8000   :-------------:           >8000   :-------------:
        : 2nd 8K      :                   : 1st 8K      :
        : chip        :                   : chip        :
        :             :                   : repeated    :
>9000   :-------------:           >9000   :-------------:
        : 1st 8K      :                   : 1st 8K      :
        : chip        :                   : chip        :
        : repeated    :                   : repeated    :
>7000   :-------------:           >7000   :-------------:
        : 1st 8K      :                   : 1st 8K       :
        : chip        :                   : chip         :
        :             :                   :              :
>5000   '-------------'           >5000   '-------------'
```
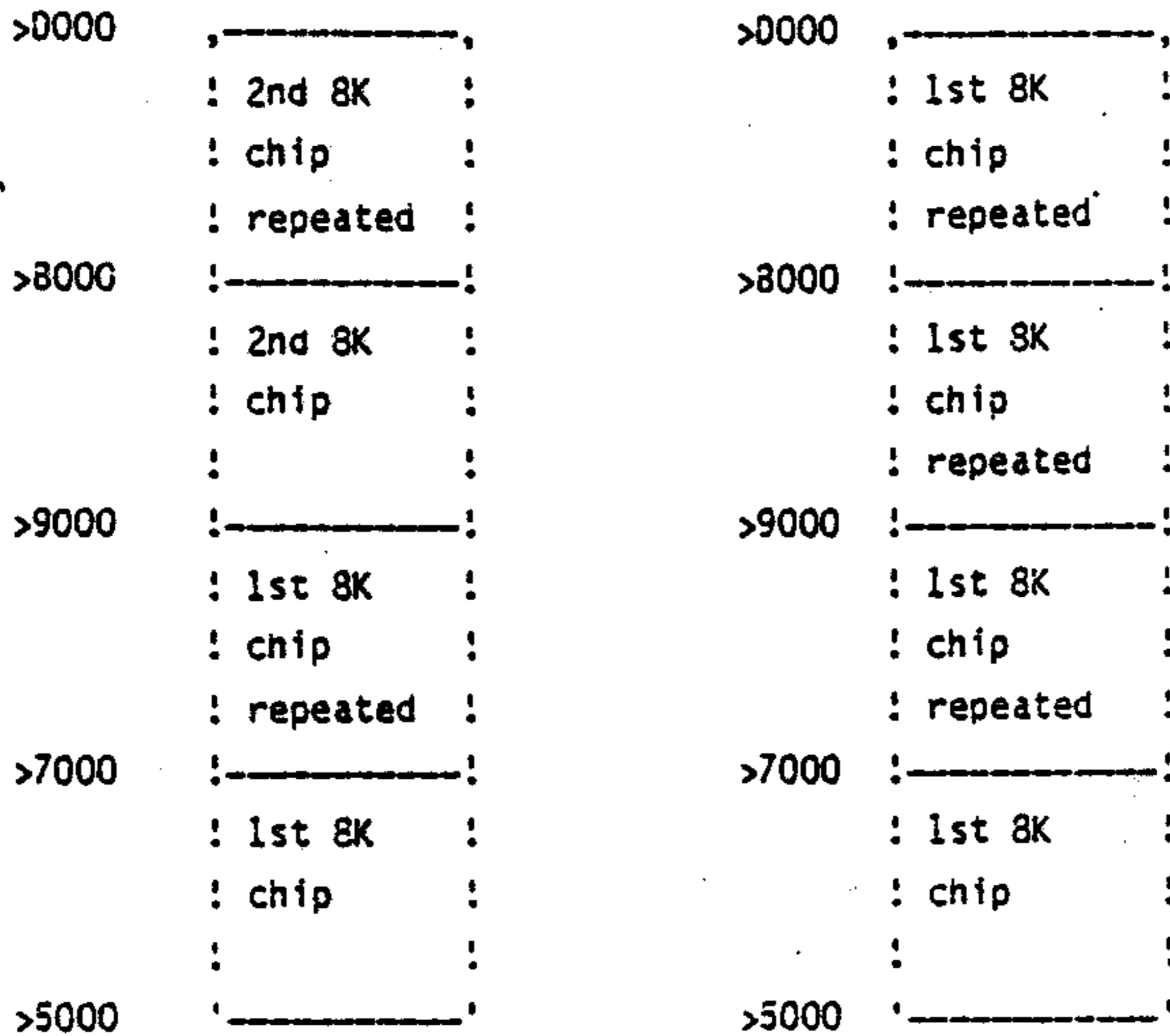
         16K CARTRIDGE RAM              8K CONSTANT MEMORY™ CARTRIDGE

With a 16K cartridge, RAM is available in two 8K blocks beginning at >5000 and
>9000.  With a Constant Memory™ cartridge, a single 8K block at >9000 is
available.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alone = pound sign; % = percent sign; c = at sign; y = question mark

The CARTLOAD Subprogram

Loading programs into a cartridge after the cartridge header (and perhaps
other programs) have been loaded requires CARTLOAD. Information in the
cartridge header and in the header of any last program already loaded is
updated by CARTLOAD. The subprogram is called from BASIC as in the following
example.

        CALL CARTLOAD("1.PROG5")

The pointer at >9004 in the cartridge header points to the header of the first
program or subprogram in the cartridge. The program header for the first
program, in turn, contains an offset which points to the header for the next
program. The program header for the last program in the cartridge contains
>0000 in the offset word. In this manner, the cartridge header and program
headers taken together contain a linked list of programs stored in the
cartridge. CARTLOAD loads in any new program or programs, relocates them
appropriately for execution at their current addresses, and patches the offset
of the (first) header in the new program(s) to replace the >0000 offset in the
header of the program which had been the last one in the list. The new
program(s) must contain >0000 in the offset of the (last) header to indicate
the new end of the list.

All files loaded by the cartridge loader must be in relocatable
multiple-record format produced by the linker M command. An attempt to load
an absolute file or a single-record relocatable file causes a Bad program type
error.

If memory is unavailable for loading a program into cartridge RAM from >9000
through >CFFF, the loader attempts to load the program in RAM from >5000
through >9000.

In addition to I/O errors, five other types of errors are identified by
messages from CARTLOAD and CARTINIT, as shown in the following table.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  ^ = percent sign;  o = at sign;  u = question mark

---

## CARTLOAD AND CARTINIT ERROR MESSAGES

Illegal syntax          Bad syntax in the assembly language call

Can't do that           Missing cartridge header

Bad program             The input file is not a multiple-record
type                    relocatable file

No RAM in               There is no RAM in the cartridge
cartridge

Memory full             Insufficient space in system RAM for the relocation
                        table or in the cartridge for the program


CARTRIDGE-SAVE PROGRAM (MEMSAVE)
The cartridge-save utility MEMSAVE saves a block of memory as a
multiple-record absolute file for later loading by the absolute loader. All
of the programs which are loaded into a RAM cartridge can be saved as a block
with beginning and ending addresses provided in the MEMSAVE command. Blocks
containing code from ROM or the register-file area cannot be saved by MEMSAVE.

Like ABSLOAD and CARTLOAD, MEMSAVE cannot be used to save data from cartridge
RAM while the Editor/Assembler cartridge is in the CC-40. Downloading MEMSAVE
follows the procedures used in downloading the two loaders (see "Saving and
Loading ABSLOAD and CARTLOAD," above).

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed

alpha = pound sign;  b = percent sign;  c = at sign;  t = question mark

When MEMSAVE has been loaded by BASIC with the CALL LOAD subprogram, the program is run with a CALL command which specifies a filename, a beginning address, an ending address, and a record length.  If a record length is not included, the file is saved as a single record.  The following command, for example, saves the 8-kilobyte block from >9000 through >AFFF in records of 256 bytes.

    CALL MEMSAVE("1.HICART",">9000,>AFFF",256)

Address values are entered by hexadecimal numbers within quotes and separated by commas.  The ">" sign is optional.  Record lengths are in decimal.

In addition to I/O errors, four other types of errors are identified by messages from MEMSAVE, as shown in the following table.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

## MEMSAVE ERROR MESSAGES

Illegal syntax          Bad syntax in the assembly language call

Can't do that           Attempt to save block from ROM memory or the
                        register-file  area (>0000 to >0200) to a mass-storage
                        file; also attempt to save block which does not
                        contain RAM

Bad value               1. Record length specified in filename is either
                        negative or greater than 32767
                        2. Non-hex characters were used in addresses provided
                        with the MEMSAVE command
                        3. Hexadecimal value in an address is greater than
                        >FFFF

Bad argument            The starting address is greater than the ending address


## FILE CONVERSION AND TRANSFER UTILITIES

The following utility programs are provided to convert between files with ALDS
format and files of other formats (BASIC, 990) and to transfer files to the
CC-40 from external computers.

!o! Multiple-record to single-record relocatable files

!o! List file to ALDS text file

!o! External-computer text files to ALDS text files

!o! 990 tagged-object to ALDS tagged-object files

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  c = at sign;  μ = question mark

---

    !c! BASIC program image to ALDS text file

    !c! ALDS test file to BASIC program image

## Multiple- To Single-Record File Conversion Utility (CONVERT)

The utility program called CONVERT reformats a file saved as multiple records
of relocatable object code into a single-record relocatable file.
Multiple-record files can, after this conversion, be loaded and run directly
under BASIC.

The conversion program is accessed from the ALDS menu by typing [C].  No
prompt appears in the ALDS menu for CONVERT.

The prompt Input file requests the name of the multiple-record relocatable
object file to be converted.  The prompt Output file then requests the name of
the single-record object file to be written.

In addition to I/O errors, CONVERT identifies two other types of errors by the
messages shown in the table below.

### CONVERT ERROR MESSAGES

Bad program type       The input file is not a multiple-record relocatable
                       object-code file

Length error           The input file is of incorrect type or it contains a
                       header with incorrect block size

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  υ = question mark

## File Transfer Utility (TRANSFER)

The utility program called TRANSFER permits the CC-40 to perform three file
transfer operations. It can input a list file with records of 80 or fewer
bytes from one HEX-BUS<sup>TM</sup> mass-storage device and write it to a file on
another HEX-BUS device. A list file contains lines of printable ASCII text
without end-of-line bytes.

The TRANSFER utility can also input a file from an external computer and
convert it to an ALOS text file on a mass-storage device. In addition, it can
input files containing TI-990 tagged-object code and convert them to ALOS
tagged-object format.

## Text Transfer from One Device to Another

The transfer utility program is accessed by pressing [T] in response to
T)ransfer from the ALOS menu.

The message S)ource O)bject prompts for input to determine whether a standard
text or 990 tagged-object code is to be received.

Next, TRANSFER issues the prompt C)runch U)ncrunch. If [C] is pressed, the
file to be output is set to be in crunched (tokenized) form. If [U] is
pressed, the file is output in the same form in which it is input.

When the input file prompt appears, the name of the file to be transferred is
entered.

The Handshake (Y/N)υ prompt appears to permit using communications protocol
with computers external to the CC-40. For CC-40 device-to-device transfer,
[N] is pressed to use internal HEX-BUS<sup>TM</sup> protocol.

Next TRANSFER displays an Output file prompt. The name of the file to receive
the text is entered. An optional record length can be specified following the
name by typing a space and R=NNNN.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  = percent sign;  = at sign;  = question mark

The message Transferring... appears in the display until the file copying is
complete and the ALDS menu returns to the display.


Text Transfer from an External Computer

The computer from which the file is being transferred must be programmed to
send the file with the format and protocol required by TRANSFER. The listing
shown below provides an example of such a program written in BASIC. The
program uses handshaking (line 150) to determine when to send the next line;
no echo is provided to the display. An asterisk (*) is sent by TRANSFER each
time it is ready to receive an 80-character line. An end of the file is
recognized by TRANSFER when it receives an ASCII >7F or 127 (DEL).

```
100 OPEN α2:"RS232.BA=4300.EC"    . Open RS232, no echo
110 INPUT "File download utility Enter file name: ":NAME$
130 LINPUT α1:TXT$                . Get line from source file
140 PRINT α2:SEG$(TXT$,1,80);     . Transfer first 80 characters
150 INPUT α2:HSK$                 . Get clear to send character (handshake)
160 IF EOF(1)=0 THEN 130          . If not end of file, continue
170 PRINT α2:CHR$(127)            . Signal end of transmission
180 CLOSE α1                      . Close source
190 CLOSE α2                      . Close RS232
```

For transfer of files external to the CC-40, the prompts through C)runch
U)ncrunch are answered as they are for CC-40 internal file-to-file copying.
When the prompt Input file appears, however, the number of the device (and
characteristics such as RS232 transmission speed) through which the text is to
be transferred must be provided.


The prompt Handshake (Y/N) appears. When [Y] is pressed, TRANSFER is set to
send a "*" character to the external computer when it is ready to receive
text. Without handshaking, data is lost during transfer operations.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

Next, the prompt Echou (Y/N)u appears. Responding [Y] to the prompt causes
the CC-40 to expect the external computer to echo the handshaking character
upon receipt.

When TRANSFER displays an Output file prompt, the name of the file in which
received text is to be stored is entered.

With TRANSFER running, the message Ready... must appear in the display before
the external computer begins to transmit. With reception of the first
character from the external computer, the message Transferring... closes the
file and the ALDS menu returns to the display.

TI-990 Tagged-Object File Transfer

When TRANSFER is to receive TI-990 tagged-object code files from an external
computer, all prompts except one are answered as above. The S)ource O)bject
prompt is answered by pressing [O].

After 990 code is received and before TRANSFER writes it to the output file,
the code is converted to ALDS format tagged-object code.

The table below shows the correspondence between 990 tagged-object format and
the tagged-object format output by the ALDS assembler.

The primary difference between 990 and ALDS format is that the 990 assembler
always outputs uncompressed tagged-object code while the ALDS assembler always
outputs compressed tagged-object code. Code in ALDS format has bytes and
words represented directly in binary values, while code in 990 format has
bytes and words represented in ASCII characters. In ALDS format, a byte of
data with a value of 32 is represented in a two-byte field containing an ASCII
"H" tag and a binary >20 value. In 990 format, the same byte is represented
in a three-byte field containing an ASCII "*" tag, an ASCII "2," and an ASCII
"0."

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
alone = pound sign;  ▵ = percent sign;  ¢ = at sign;  � = question mark

Most tags, although designated by different characters, have similar functions in both formats.  Tags which do not have the same functions are noted with a ".." in the table.

Differences in the content of files are primarily in the expression of symbols expressing assembled DEF and REF directives.  Both DEFed and REFed symbols from the 990 linker are fixed-length six-character symbols with spaces to the right of symbols containing fewer than six characters.  The ALDS assembler produces symbols which are one to eight characters long with a length byte preceding the symbol.  The length byte has a value of one less than the number of characters in the symbol.  The 990 REFs also contain an unused word that is not stored in CC-40 linker format.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = oercent sign;  σ = at sign;  ɥ = question mark

CORRESPONDENCE 990 AND ALDS TAGGED OBJECT CODE

| 990 Tag | ALDS Tag | Value | Description |
|---|---|---|---|
| :. | A | Word Program ID | Cartridge indentifier—Compressed |
| | | | Word = Highest relocatable address |
| K | .. | Word Program ID | Cartridge indentifier—Uncompressed |
| | | | Word = Highest relocatable address |
| 0 | .. | Word Program ID | Cartridge indentifier—Uncompressed |
| | | | Word = Highest relocatable address |
| .. | B | Reserved | Reserved tag |
| 9 | C | Word | Absolute load address |
| A | D | Word | Relocatable load address |
| .. | E | Byte Symbol | REFed symbol |
| | | | byte = symbol length – 1 |
| 3 | .. | Word 990-symbol | REFed symbol (Ignore Word) |
| 4 | .. | Word 990-symbol | REFed symbol (Ignore Word) |
| .. | F | Byte Word Symbol | Absolute DEFed symbol |
| | | | Word = symbol value, |
| | | | Byte = symbol length – 1 |
| 5 | .. | Word 990-symbol | Absolute DEFed symbol |
| | | | Word = symbol value |

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  % = percent sign;  @ = at sign;  u = question mark

| 990 Tag | ALDS Tag | Value | Description |
|---------|----------|-------|-------------|
| .. | G | Byte word Symbol | Relocatable DEFed symbol |
|    |   |                 | Word = symbol value, |
|    |   |                 | Byte = symbol length – 1 |
| 5  | .. | Word 990-symbol | Relocatable DEFed symbol |
|    |   |                 | Word = symbol value |
| * | H | Byte | Absolute byte of data |
| B | I | Word | Absolute data |
| C | J | Word | Relocatable data |
| E | K | Word1 Word2 | External reference |
|   |   |             | Word1 = Reference number |
|   |   |             | Word2 = Value added to reference |
| 7 | L | Word | Checksum |
| 8 | M | Word | Ignore checksum |
| F | N |  | End of record |
| : | O |  | End of file |

Value definitions

Word = 16 bit value          Symbol      = 1..8 ASCII characters
Byte = 8 bit value           Program ID = 8 ASCII characters
                             990-symbol = 6 ASCII characters

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

Transfer Utility Errors

In addition to I/O errors, TRANSFER identifies four other errors with messages
listed in the following table.

## TRANSFER ERROR MESSAGES

| | |
|---|---|
| Invalid module | The first tag in a 990 file is not a "K" or "0" (IDT) tag |
| Duplicate module | A second IDT tag has been found |
| Invalid value | An ASCII character which is not a valid hexadecimal digit is present in a data field |
| Invalid tag | An ASCII character which is not a valid 990 tag appears where a valid one should appear |

## Basic to ALDS Text Utilities (SAVEA and COPYA)

Two utility programs called SAVEA and COPYA permit using the ALDS screen
editor to edit BASIC program files.

For saving a program currently in BASIC program memory in ALDS format, SAVEA
is used.  From BASIC, with the ALDS cartridge in the CC-40, SAVEA produces a
file named "BASPROG" on device 1 in 128-byte records when the following
command is entered.

    CALL SAVEA("1.BASPROG",128)

The SAVEA utility cannot be called from within a BASIC program.  The name of
the file must be typed in double quotes, and the filename and optional record
length must be enclosed in parentheses.  If no record length is specified,
SAVEA selects a length as great as free memory permits.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign:  % = percent sign:  : = at sign:  y = question mark

The COPYA utility permits BASIC to copy an ALDS text file into memory in BASIC
format.  The ALDS-format "BASPROG" file from device 1, for example, is read
into program memory by BASIC with the following command.

        CALL COPYA("1.BASPROG")

The COPYA utility cannot be called from within a BASIC program.  The name of
the file must be entered inside double quotes which are enclosed in
parentheses.

In addition to I/O errors, errors occurring during SAVEA and COPYA cause the
messages listed in the following table to appear.

### SAVEA AND COPYA ERROR MESSAGES

| | |
|---|---|
| Illegal syntax | Bad syntax in the assembly language call |
| Protection violation | Attempt to save a protected BASIC program (SAVEA only) |
| Illegal in program | SAVEA or COPYA is called from within a program |
| Bad value | The specified record length is either negative or greater than 32767 |
| Bad program type | Attempt to load other than an uncrunched ALDS text file |

ALDS FILE-TRANSFER AND CONVERSION SUMMARY

The following figure summarizes the transfer of files among various ALDS
programs and external computers.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign; ▵ = percent sign; ∂ = at sign; ų = question mark

ALDS text format consists of one or more lines per record with a maximum of 74
characters in a line and each line terminated by a zero.

External text format consists of single-line records with the length of record
length fixed at 80 characters (using trailing spaces as necessary).

FLOWCHART

```
,--------------.      ,--------------.      ,--------------.
| External     |      |   BASIC      |      |  E)ditor     |
| Text file    |      |   program    |      | L)ine S)creen|
'--------------'      '--------------'      '--------------'
        |                    |                     |
,--------------.      ,--------------.              |
| - Optional - |      | BASIC -> ALDS |             |
| - RS/232   - |      | utility (SAVEA)|            |
'--------------'      '--------------'              |
        |                    |                      |
        |                    '----------------------'
        |                    |
,--------------.      ,--------------.      ,--------------.
|  TRANSFER    |      |   ALDS       |      | ALDS -> BASIC|
|  S)ource     |------|   Text file  |------| utility (COPYA)|
'--------------'      '--------------'      '--------------'
        |                    |                     |
,--------------.      ,--------------.      ,--------------.
| 990 tagged   |      | A)ssembler   |      |   BASIC      |
| object file  |      |              |      |   program    |
'--------------'      '--------------'      '--------------'
        |                    |
,--------------.      ,--------------.
|  TRANSFER    |      | ALDS tagged  |
|  O)bject     |------| object file  |
'--------------'      '--------------'
        |                    |
,--------------.      ,--------------.
| Linker       |      | L)inker      |
| control file |------|              |
| or keyboard  |      |              |
'--------------'      '--------------'
                             |
        ,--------------------+--------------------.
        |                                         |
,--------------------.              ,--------------------.
| Single-record      |              | Multiple-record    |
| Memory image file  |              | Memory image file  |
'--------------------'              '--------------------'
        |                                   |
   ,----+----.                        ,-----+-----.
   |         |                        |           |
,----------. ,----------.      ,----------. ,------------.
|Relocatable| | Absolute |      | Absolute | | Relocatable|
'----------' '----------'      '----------' '------------'
        |         |                  |           |
        |     ,----------------------'           |
        | ,--------------.                        |
        '-| CONVERT      |<-----------------------'
          '--------------'
        |         |                              |
,--------------. ,--------------.      ,--------------.
| "CALL LOAD"  | |  Absolute    |      |  Cartridge   |
| "OLD" / "RUN"| |  Loader      |      |  Loader      |
'--------------' '--------------'      '--------------'
        |               |                      |
,--------------. ,--------------.      ,--------------.
|    RAM       | | RAM / Cart. RAM|    | Cartridge RAM|
'--------------' '--------------'      '--------------'
                        |                      |
                ,--------------.      ,----------------.
                | Memory Save  |      | Absolute Memory|
                | Utility      |----->| Image file     |
                '--------------'      '----------------'
```

Summary of ALDS File Conversion and Transfer

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
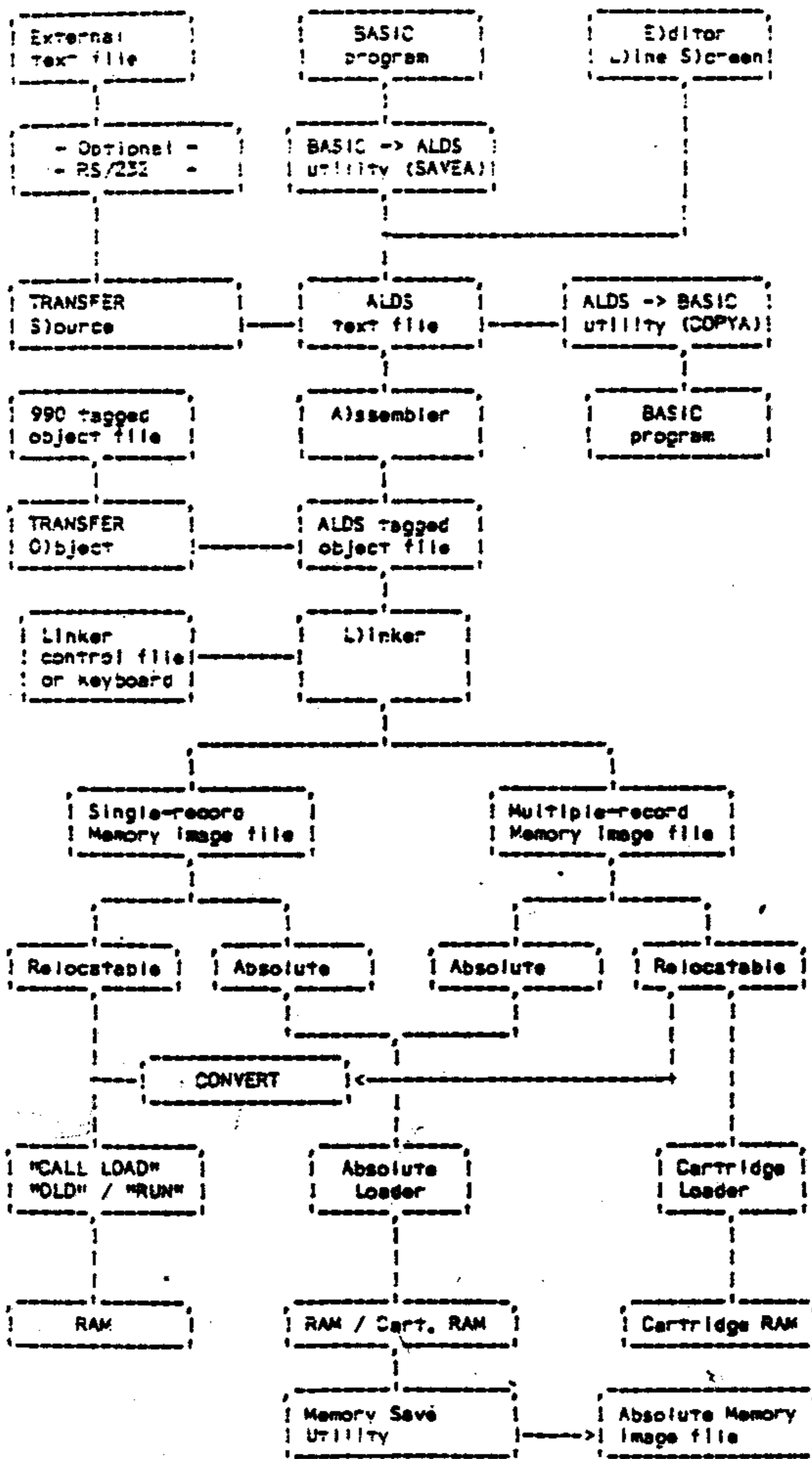alpha = pound sign;  A = percent sign;  o = at sign:  u = question mark

## CHAPTER 8
## DEBUG MONITOR

This chapter describes the DEBUG Monitor and gives an example of the use of the monitor to alter and run a machine-language program.

The DEBUG Monitor allows examination, modification, and controlled execution of machine-language programs which have been developed by the ALDS.

The monitor permits displaying and modifying data in memory, moving blocks of data in memory, executing programs until preset breakpoints are reached, and executing programs in steps of single machine-code commands.

The BASIC and DEBUG Monitor operating environments (RAM locations) are compatible. An ALDS-developed program called from a BASIC program, for example, can have breakpoints set by DEBUG; when the BASIC program is run and CC-40 control is transferred to the machine-language subroutine, the breakpoints cause the monitor to be entered. All elements supporting BASIC program execution remain intact, and the machine-language and BASIC programs can resume when use of the monitor is complete.

## RUNNING THE DEBUG MONITOR
The DEBUG Monitor is entered by a CALL DEBUG BASIC interpreter command or a CALL DEBUG statement line in a BASIC program. The monitor prompts with a colon (:) for single-letter commands and appropriate values.

## MONITOR COMMANDS
Keyboard-entered command lines which control DEBUG Monitor operation consist of a single-character uppercase or lowercase alphabetic command followed by any required numeric values and [[ENTER]] or [[SPACE BAR]]. Multiple values are separated by a single space.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;   Δ = percent sign;   σ = at sign;   υ = question mark

Command values are expressed in hexadecimal or decimal numbers.  Decimal
values must be preceded by a period; numbers without a period in front of them
are evaluated as hexadecimal values. The symbol for hexadecimal value is not
used in the DEBUG Monitor.  Values greater than 65535 or >FFFF are truncated
to the left.  All displayed values are in hexadecimal form.

For command-line editing, the [Left Arrow] and [Right Arrow] keys position the
cursor over any character entered except a period.  At any time during
command-line entry, the [[BREAK]] key can be pressed to clear the line and
return to monitor command level (with the ":" prompt displayed).  When the
DEBUG Monitor encounters an error in a command line, it clears the display and
prompts again with WHAT:.

Display—Memory Command: D
The display—memory command causes the content of eight bytes of memory
beginning at the starting address expressed in the command to appear in the
display.  Display of the contents of subsequent eight-byte blocks of memory is
caused by pressing [+] or [[SPACE BAR]].  Display of the contents of previous
eight-byte blocks of memory is caused by [-].  Further display terminates when
[[ENTER]] is pressed.

Because 7000-family data registers are in memory, the display—memory command
can be used to display registers in the general purpose file (>0000 through
>007F) or the peripheral file (>0100 through >01FF).

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  = = at sign;  y = question mark

The syntax of the display memory command is shown below.


Command: D


Enter:    D NNNN


NOTE:  Values may be entered in either decimal or hexadecimal. Entering D
without a value is equivalent to D 0000.


Memory-Modify Command: M
The memory-modify command permits display and keyboard entry of single-byte
memory contents.


The value stored at the address specified in the command is displayed. (The
default for the modify-memory command starting address is >0000, the A
register.)  If a new value is entered, the value replaces the value
displayed.  If [+] or [[SPACE BAR]] is pressed following display of the memory
contents or following entry of a replacement value, the contents of the
subsequent address are displayed.  If [-] is pressed at either time, the
contents of the previous address are displayed.  The command terminates when
[[ENTER]] is pressed at either time.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

Contents of the registers can be easily examined and modified by the
memory-modify command because they are addressed as memory registers.
General-purpose file registers 0 through 127 are addressed as memory locations
>0000 through >007F, and peripheral file registers 0 through 255 are addressed
as locations >0100 through >01FF.

The syntax of the memory-modify command is shown below.

Command: M

Enter:    M NNNN

NOTE:  Values can be entered in decimal and hexadecimal. Entering M without a
value is equivalent to M 0000.

Processor-Register Modify Command: P
The processor register modify command permits the three registers in the
processor, the program counter (PC), status register (ST), and stack pointer
(SP), to be displayed and modified in much the same manner in which the
contents memory locations are modified.  The modification is effective with
the next E (execute) or S (single-step) command.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  ^ = percent sign;  @ = at sign;  u = question mark

After [P] has been pressed, the value stored in the current DEBUG Monitor
program counter (PC) is displayed.  If a value is then typed in at the
keyboard, it becomes the new DEBUG Monitor PC, which specifies the point in
the program at which the program begins execution with the next execute (E) or
single-step (S) command.  If [[SPACE BAR]] is pressed in response to the PC
display or in termination of the  new PC value, the contents of the status
register (ST) are displayed.  If a value is typed, it becomes the new ST after
the next E or S command; pressing [[SPACE BAR]] or terminating the new ST
value with [[SPACE BAR]] causes the stack pointer (SP) to be displayed.   If a
value is entered it becomes the new SP with the next E command.

Pressing [[ENTER]] or terminating any new value with [[ENTER]] at any time
terminates the command (but retains any values entered).

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign:  μ = question mark

The modify-register command can display and prompt for all three register
values, as shown below.


                         Command: P           .



               Press:                   P


               Displays:                PC=MMMM
                                        ST=PP
                                        SP=RR


NOTE:  Proceed from one display to the next by pressing [[SPACE BAR]] or exit
from the P command by pressing [[ENTER]]. To alter any of the values in the
displayed registers, type the new value and press either [[ENTER]] or
[[SPACEBAR]]. Values may be entered in either decimal or hexadecimal.


Copy-Memory Command: C
The copy-memory command causes the contents of one block of memory to be moved
to another block.  One byte is moved and the source and destination addresses
are incremented by one. The single-byte moves continue for the length
specified in the command.  If the destination address is less than or equal to
the source address plus the length of the move, blocks can overlap.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  a = percent sign;  c = at sign;  u = question mark

The overlap feature of the copy-memory command can effect a "fill memory"
operation. When an initial "fill" character is stored by a modify-memory
command (M) at the beginning of a source block and the start of the
destination block for the move is set one byte higher, the whole block is
filled with the character.

The syntax of the copy memory command is shown below.

### Command: C

### Enter: C NNNN MMMM PPPP

NOTE: NNNN-PPPP may be entered in either decimal or hexadecimal. NNNNN
specifies the address of the first byte to move. MMMM specifies the address
into which the first byte is moved. PPPP specifies the number of bytes to move.

## Execute Command: E

The execute command transfers processor control to the program location
specified by the DEBUG Monitor program counter (set by the P command,
described above).

The E command must be followed by [[ENTER]]. If [E] is pressed accidentally
and [[ENTER]] has not been pressed, pressing [[BREAK]] causes the command to
be ignored.

Any new system- or cartridge-memory page selection (see R command, below) or
new status register or stack-pointer value takes effect with the the E command.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;   Δ = percent sign;   σ = at sign;   u = question mark

The syntax of the execute command is shown below.


                              Command: E


                               Enter: E


NOTE: No parameters are used with the execute command.  The address at which
execution begins is the address in the DEBUG Monitor program counter.


## Breakpoint-Modify Command: B

The breakpoint-modify command, in a manner like the memory-modify and
register-modify commands, displays the current breakpoints and permits the
assignment of new addresses to them.  Breakpoints are addresses at which
program execution is halted so that the DEBUG Monitor commands can be used to
examine memory and register contents at that point in the program.
Breakpoints can only be used with programs which are stored in RAM, and they
should not be set at the current program-counter value.

When the DEBUG Monitor is reentered after a program reaches a breakpoint, both
breakpoints are turned off.

Breakpoints are set for particular addresses by the breakpoint-modify
command.  After being set, either of the two breakpoints can be turned off by
entering >0000 for an address in the breakpoint-modify command.

A breakpoint remains in effect until it is cleared with a breakpoint-modify
command or until the processor "executes" it.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alone = pound sign;  A = percent sign;  c = at sign;  v = question mark

The syntax and procedure used with the breakpoint-modify command are shown
below.


                          Command: B



              Press:              B

              Displays:           B MMMM
                                  PPPP


NOTE: MMMM and PPPP are addresses at which program execution is to be halted
for memory and register examination and modification. Proceed from one display
to the next by pressing [[SPACE BAR]] or exit from the B command by pressing
[[ENTER]]. To alter any of the displayed breakpoints, type the new value and
press either [[ENTER]] or [[SPACEBAR]]. Values may be entered in either
decimal or hexadecimal.


ROM-Page Modify Command: R
The ROM-page modify command permits setting the cartridge (addresses >5000
through >CFFF) page and the system ROM (addresses >D000 through >FFFF) page to
any page from 0 through 3.

Pressing [[SPACE BAR]] after display or modification of the cartridge page
causes the system-ROM paging prompt to be displayed.  Pressing [[ENTER]]
terminates the page-modify command without display of the system-ROM
page-modify prompt.

underlining in text (except heads) indicates a typestyle other than normal text; single brackets indicates information to be typed; double brackets indicates key to be pressed
alpha = pound sign;  ∆ = percent sign;  σ = at sign;  u = question mark

The cartridge page-modify command takes effect immediately. The DEBUG Monitor commands used to display ROM or RAM, modify RAM, set breakpoints in RAM, and other functions use the page which has been set. The system-ROM page-modify command takes place only after an execute (E) command is entered. All commands except E reference the page from which the DEBUG Monitor runs, page 3. Displays of system ROM, are, for example, always from page 3. An address in system ROM at which execution is to begin, however, is on the page last selected.

The syntax and procedures used with the page-modify command are shown below.

Command: R

Press:              R

Displays:           CARTRIDGE PAGE=M
                    SYSTEM PAGE=P

NOTE:  Proceed from one display to the next by pressing [[SPACE BAR]] or exit from the R command by pressing [[ENTER]]. To alter any of the displayed page numbers, type the new page number and press either [[ENTER]] or [[SPACEBAR]]. Page numbers are within the range 0-3. The cartridge page number displayed (M) is currently in effect, but the system page number displayed (P) is only in effect upon exit from the DEBUG Monitor.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  % = percent sign;  @ = at sign:  ? = question mark

## Single-Step Command: S

The single-step command causes only the single processor command at the
location in the program counter (PC) to be executed.  After command execution,
program control returns to the monitor for subsequent examination or
modification of registers and memory, single-stepping, and so on.  The effect
of the single-step command is as if a breakpoint were set at the command
following the one pointed to by the DEBUG Monitor program counter.  One
consequence of the way the single-step command operates is that an instruction
which jumps or branches to itself cannot be single-stepped.  Another effect is
that single-step operation cannot be used with programs in ROM or during calls
to subroutines in ROM.

Execution of a single-step command turns off both breakpoints.

Single-step command syntax is shown below.


### Command: S


### Enter: S


NOTE: No parameters are used with the single-step command.  The address of the
instruction executed is the address in the DEBUG Monitor program counter.

## The Quit Command: Q

The quit command causes the CC-40 to exit the DEBUG Monitor and reenter BASIC
in one of two ways.  The BASIC program being executed when the CALL DEBUG
instruction was encountered can be resumed with a QB instruction, or BASIC can
be restarted at the command-interpreter level with a QI instruction.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  u = question mark

The syntax of the quit command is shown below.


                        Command: Q


                Entered:     QI or QB


NOTE: QI returns the CC-40 to the command level of the BASIC interpreter, and
QB returns the CC-40 to any program running under BASIC.


The Help Command: u
The help command displays all of the command codes used by the DEBUG Monitor.
Use the help command as shown below.


                        Command: u


                        Entered: u


        Displayed: COMMANDS= Q,B,E,M,C,S,D,P,R


NOTE: The help command displays the codes until [[ENTER]] is pressed.



DEBUG MONITOR OPERATION CONCEPTS
When the DEBUG Monitor is called from BASIC by a CALL DEBUG command or program
statement, it sets up the last 80 bytes of the BASIC program-line crunch
buffer (>0900 through >094F) for temporary storage:  anything stored in this
area is therefore overwritten.  The colon (:) is the display prompt while the
monitor is active.


Setting a breakpoint in a program location causes the DEBUG Monitor to save
the command and to place a single-byte TRAP 11 command (command code >F4) at
the location specified.  (The bytes replaced by TRAP 11 commands for both
breakpoints and their addresses are stored in >08F4 through >08F9.)

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alone = pound sign;  Δ = percent sign;  c = at sign;  u = question mark
_____

The program containing the breakpoint executes all machine-language commands
until the TRAP 11 is read. The TRAP 11 command causes the DEBUG Monitor to be
reentered at a point at which both breakpoints are cleared and the original
command is restored to its memory location.

Storing a TRAP 11 in memory through a modify-memory instruction has the same
effect as setting a breakpoint, with one very useful difference: the
directly-stored TRAP 11 is not cleared upon return to the monitor.  The
breakpoint remains set for future use after control is transferred to the
monitor.

The following events occur when a breakpoint is encountered.

1.  The program counter, current status, registers A and B, current ROM
    page, and 2 reserved bytes are pushed onto the stack.  The stack used
    for saving these values is the stack set by the program being
    executed; therefore, the design of programs to be debugged with
    breakpoints or single stepped must account for eight additional bytes
    being pushed onto the stack.

2.  R0-R53 and R83-R103 are saved in the crunch buffer (along with five
    other bytes of temporary data).

3.  Both breakpoints are cleared.

4.  The program counter, status register, and stack pointer are displayed
    (followed by a colon as the prompt which indicates return to the DEBUG
    Monitor).

Breakpoints remain active (as long as either or both are not equal to 0) when
BASIC is reentered from the DEBUG Monitor.  When the machine-language program
is executed by calling it from BASIC and a breakpoint is reached, the monitor
is called.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;   Δ = percent sign;   σ = at sign;   u = question mark

Breakpoints are not cleared when the DEBUG Monitor is reentered from BASIC
with a CALL DEBUG.  Breakpoints are, however, cleared when the single—step
command is entered.  When the CC-40 is initialized, breakpoints are cleared.

## DEBUG MONITOR USE WITH AN ALDS—DEVELOPED PROGRAM

The "BEEP" program developed in chapter 1 of this guide contains no
relocatable addresses in the executable code.  It can therefore be entered for
execution at any memory location that does not destroy BASIC or DEBUG Monitor
RAM.  The figure below shows a memory image of the executable code in "BEEP"
with addresses beginning at >1000.

| ADDRESS | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1000 | 88 | 04 | 00 | 5D | A2 | 01 | 15 | 52 | 1D | CA | FE | A2 | 00 | 15 | 52 | 20 |
| 1010 | CA | FE | 0B | 5D | E3 | EE | 0A | | | | | | | | | |

### Entering the Machine—Language Program

The following steps show how to use the DEBUG Monitor to enter and display the
"BEEP" machine—language program.

STEP 1:   In response to the BASIC command prompt, type in [CALL DEBUG].

STEP 2:   When the MONITOR: prompt appears type in the modify—memory
command [M 1000] (the monitor inserts the space after "M").

STEP 3:   Type in each of the 23 bytes of the machine—language program
exactly as they appear above.  After each byte until the last
is typed, press [[SPACE BAR]].  After the last byte is typed,
press [[ENTER]] to return to the DEBUG monitor command level
with the ":" prompt to the display.

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
aicha = pound sign;   A = percent sign;   o = at sign;   y = question mark

STEP 4:    Type in [D 1000] to display the first eight bytes of "BEEP,"
           and verify each byte with the commands listed above Note the
           values and memory locations of any incorrectly-entered bytes.

STEP 5:    Press [+] to display the second eight bytes for verification.
           Again note the locations of incorrect bytes.

STEP 6:    Press [+] again, and verify the first seven of the third
           eight-byte display.

STEP 7:    With verification done, press [[ENTER]] to return to the ":"
           prompt.

STEP 8:    If any incorrect bytes have been found, use the addresses of
           these bytes with the M command to correct them.

The machine code "BEEP" program is now entered into memory.


Testing the "BEEP" Program
The "BEEP" program is executed by setting the program counter in the
7000-family processor to the entry point of the program (>1000).  When the
program executes the return from subrouting (RETS) command (>0A in the
memory-image dump), program control is returned to BASIC.  When a breakpoint
is set at >1016, control returns to the DEBUG Monitor.  The following steps
show how to execute the "BEEP" program and return control to both BASIC and
the DEBUG Monitor.

STEP 1:    Press [P] to modify the program counter in the processor. The
           display shows the current value of the counter in the form
           PC=NNNN.

STEP 2:    Type 1000 to set the program counter to >1000, the entry point
           of "BEEP."

underlining in text (except heads) indicates a typestyle other than normal
text; single brackets indicates information to be typed; double brackets
indicates key to be pressed
alpha = pound sign;  Δ = percent sign;  σ = at sign;  μ = question mark

STEP 3:    Press [E] to execute the program and return to BASIC.

STEP 4:    After the beeper has sounded and the BASIC cursor appears, type
           in the [CALL DEBUG] command to once more enter the monitor.

STEP 5:    When the MONITOR: prompt appears, press [P] and enter 1000, as
           in step 2, to once again set the program counter to the start of
           the "BEEP" program.

STEP 6:    Type [B] and 1016 to set a single breakpoint at >1016.

STEP 7:    Execute the program by pressing [E].

STEP 8:    After the beeper sounds, the breakpoint address, the value in
           the status register, and the value in the stack pointer appear
           in the display followed by the ":" prompt.


Modifying "BEEP" Program Machine Code

The sound emitted by the beeper is controlled by values entered into registers
(Review figure 5-1 for exact details of how the program works). The registers
are decremented to zero to establish the duration of the sound and the period
of the electrical impulses sent to or withheld from the beeper to establish
the frequency of the sound produced.

In the "BEEP"-program machine language commands listed above, the value
appearing in >1001 and >1002 is the immediate value loaded into the two
registers that control the duration of the sound. The value at >1008 controls
the period electrical current is sent to the beeper, and the value at >100F
controls the period in which no current is sent.

Changing the content of >1001 and >1002 to lower values shortens the tone of
the "beep."  Changing them to higher values lengthens the sound. Changing the
>0400 value in these locations to >0080 produces a short, crisp beep.
Changing the values to >FFFF causes the tone produced to last longer than 30
seconds.

Changing the contents of the immediate values at >1008 and >100F to lower
values raises the tone of the sound, and increasing the values lowers the
tone.  A value of >10 at each location provides a noticeable higher pitch, and
a value of >60 in each register provides a noticeably lower pitch.