Altair 680 Assembly Language Development System

Table of Contents

CHAPTER 1 GENERAL INFORMATION

1-1. INTRODUCTION

This manual describes the use of the Altair 680 Text Editor and Assembler package. Chapter 1 of this manual discusses the general characteristics of the software package. The Text Editor is discussed in Chapter 2 and the Assembler in Chapter 3. Chapter 4 gives details concerning the operating procedures of the Editor/Assembler package.

1-2. GENERAL DESCRIPTION

The Altair 680 Text Editor and Assembler facilitate the development of assembly language programs for the M6800 MPU.

The Text Editor may be used to create or modify alphanumeric text. In particular, the Editor provides an easy means to create and modify source programs for input to the 680 Assembler. This interactive Editor offers character, line, and character string manipulation commands.

The 680 Assembler is used to translate M6800 MPU source programs written in assembly code mnemonics into machine executable object code.

CHAPTER 2 ALTAIR 680 EDITOR

2-1. INTRODUCTION

The Altair 680 Text Editor may be used to create or modify alphanumeric text. In particular, the Text Editor facilitates the creation, correction, and modification of M6800 MPU source programs.

2-2. EDITOR INPUT

The Resident Editor accepts input text from:

System Reader Device.

System Console Device (terminal keyboard).

Commands to the Editor are supplied from the System Console Device.

2-3. EDITOR OUTPUT

The 680 Editor produces an output file on the System Punch Device. In addition, the Editor may be used to print selected portions of the edited text on the System Printer Device (terminal printer).

2-4. EDITOR OPERATION

The 680 Editor accepts input text from either the System Console Device or the System Reader Device and accepts edit commands from the System Console Device. During a typical edit operation, input text is transferred to the edit buffer. After editing, information in the buffer is transferred to the System Punch Device.

Edit operations may be performed on either characters or lines. The Editor assumes a character is any ASCII character. Non-printing characters such as CR and EOT are treated as characters by the Editor and can be manipulated accordingly. A line is a collection of characters delimited by Carriage Returns.

NOTE

When processing input text from the System Console Device, the Editor echoes a Line Feed with each Carriage Return entered. Text entered from the System Reader Device need not include Line Feed characters, since they will be supplied automatically following Carriage Returns in the Editor output.

Edit operations are performed on portions of the text held in an edit buffer in the 680 memory. A buffer pointer is maintained to specify a character location within the edit buffer. Certain of the edit operations are performed on lines or characters located with respect to the buffer pointer. It is convenient to think of this pointer as being located between two characters. As shown in the example below, the buffer pointer is located between the E and X of TEXT.

2-5. LOADING THE EDITOR

Two versions of the Editor are supplied. The first version, which is designed to be co-resident with the Assembler, is supplied on a tape marked

ALTAIR 680 ASSEMBLER/EDITOR

If the Assembler and Editor are to be used together, this tape should be loaded. (See Chapter 7 for details on operating procedures for the Assembler and Editor.) The second version, which is designed for stand alone operation of the Editor, is supplied on a tape marked

ALTAIR 680 EDITOR

If only the Editor is to be used, this tape should be loaded.

2-6. EDITOR INITIATION

The Editor is started by using the PROM Monitor's J command to begin execution at Ø1Ø7. Starting the Editor at Ø1Ø7 causes the Edit Buffer to be cleared. If it is necessary to re-enter the Editor for the purpose of editing text which is already contained in the Edit Buffer, execution should begin at address Ø1ØA.

2-7. EDITOR COMMANDS

The Editor prints "@" at the left margin as a prompt whenever it is waiting for a command. Commands to the Editor are single characters entered from the System Console Device. Some Editor commands have arguments associated with them. Editor commands must be terminated by two ESC ($1B) characters. (Since CR is a legal text character, it could not be used for command termination.) The two ESC characters mark the end of a command and cause the Editor to begin execution.

NOTE

Since ESC is a non-printing character, the Editor echoes "$" whenever ESC is entered.

The Editor commands are described in the following paragraphs and summarized in Table 2-1. For purposes of description, the commands are grouped into four categories.

Input/Output Operations
Buffer Pointer Operations
Edit Operations
Exit Editor

2-7.1 INPUT/OUTPUT OPERATIONS

Input/Output Operations control the transfer of information between the edit buffer and the System Reader Device, System Punch Device, and the System Printer Device.

2-7.1.1 A--APPEND

FORMAT: A

DESCRIPTION: The Append command causes input text to be transferred from the System Reader Device and appended to the Edit buffer. The transfer is terminated by one of the following conditions.

1. End of file character ($1A)
2. Workspace full

Null, rubout, LF, ESC, Backspace (Control H), Cancel (Control X), Readeron, Punchon, Readeroff, Punchoff, and EOF characters are deleted from the input text.

EXAMPLE: Assume that the Editor has been loaded into memory and is running. Also assume that a tape containing the following information has been loaded into the System Reader Device.

```
10 NAM PGM(CR) (LF)
20 OTP M MEMORY FILE OPTION(CR) (LF)
30 OPT O OUTPUT OBJECT TAPES(CR) (LF)
40 OPT S SELECT PRINTING SYMBOLS(CR) (LF)
50 ORG 8192(CR) (LF)
60 LDA B ADDR(CR) (LF)
70 COUNT EQU @8  @ INDICATES OCTAL(CR) (LF)
80 START LDS *STACK INZ STACK POINTER(CR) (LF)
90 LDX ADDR(CR) (LF)
100 LDA B *COUNT IMMEDIATE ADDRESSING(CR) (LF)
110 BACK LDA A 10 DIRECT ADDRESSING(CR) (LF)
120 CMP A 2,X INDEXED ADDRESSING(CR) (LF)
130 BEQ FOUND RELATIVE ADDRESSING(CR) (LF)
140 DEX IMPLIED ADDRESSING(CR) (LF)
150 DEC B ACCUMULATOR ONLY ADDRESSING(CR) (LF)
160 BNE BACK(CR) (LF)
170 WAI WAIT FOR INTERRUPT(CR) (LF)
180 SPC 1(CR) (LF)
190 FOUND JSR SUBRTN   JUMP TO SUBROUTINE(CR) (LF)
200 JMP START EXTENDED ADDRESSING(CR) (LF)
```

```
210 *COMMENT STATEMENT NOTE TRUNCATION 01234567890123456789(CR) (LF)
220 SUBRTN TAB COMMENT FIELD TRUNCATION 0123456789(CR) (LF)
230 ORA A BYTE  SET MOST SIGNIFICANT BIT(CR) (LF)
240 RTS  RETURN FROM SUBROUTINE(CR) (LF)
250 SPC 2(CR) (LF)
260 RMB 20 SCRATCH AREA FOR STACK(CR) (LF)
270 STACK RMB 1 START OF STACK(CR) (LF)
280 BYTE FOR $80 FORM CONSTANT BYTE(CR) (LF)
290 FCB $10,$4  $ INDICATES HEXADECIMAL(CR) (LF)
300 ADDR FDB DATA FORM CONSTANT DOUBLE BYTE(CR) (LF)
310 DATA FCC 'SET' FORM CONSTANT DATA STRING (ASCII) (CR) (LF)
320 END(CR) (LF)
330 MON(CR) (LF) (EOT)
```

A--APPEND (continued)
Entering the A command loads the contents of the tape into the edit buffer.

```
     @A$$
     @
```

NOTE

The following examples assume that the contents of the tape listed above are the only contents of the edit buffer. Remember that Line Feeds are not entered into the edit buffer. User entries in the examples will be underlined with $ indicating ESC. Notice that a minimum of two spaces are required between a line number and an instruction without a label and only one space is used between a line number and label.

2-7.1.2 E--END EDIT OPERATION

FORMAT: E

DESCRIPTION: The End command terminates the edit operation and causes the contents of the edit buffer to be transferred to the System Punch Device followed by an EOF and blank trailer. Upon completion, the Editor prints "@" and waits for further commands. The contents of the edit buffer are still available for editing. The buffer pointer remains at its previous position. The E command does not cause leader to be punched.

2-7.1.3 F--TAPE LEADER/TRAILER

FORMAT: F

DESCRIPTION: The Tape Leader/Trailer command writes fifty NULL characters to the System Punch Device. This command may be used to produce Leader/Trailer for paper tapes.

2-7.1.4 P--PUNCH

FORMAT: nP

n is a positive decimal integer less than 256. If omitted, the value is assumed to be 1.

DESCRIPTION: The Punch command causes a specified number of lines (n), beginning with the line specified by the buffer pointer, to be written from the edit buffer to the System Punch Device. The lines are deleted from the edit buffer. If a negative number n is entered, the negative sign is ignored and a positive n number of lines are punched.

2-7.1.5 T--TYPE

FORMAT: nT

n is a decimal integer in the range $-254 < n < 255$. If omitted, the value is assumed to be 1.

DESCRIPTION: The type command causes a specified number of lines (n) to be printed on the System Console Device. If n is positive, the first line typed is the current line (the line indicated by the current position of the buffer pointer). If n is negative, the n lines preceding the current line are typed. If fewer than n lines exist following (preceding) the end (beginning) of the edit buffer, only these lines are typed.

T--TYPE (continued)

EXAMPLE:

```
@5T$$
40 OPT S SELECT PRINTING SYMBOLS
50 ORG 8192
60 LDA B ADDR
70 COUNT EQU @8  @ INDICATES OCTAL
80 START LDS #STACK INZ STACK POINTER

@0T$$

@-10T$$
10 NAM PGM
20 OTP M MEMORY FILE OPTION
30 OPT O OUTPUT OBJECT TAPES

@T$$
40  OPT S SELECT PRINTING SYMBOLS

@
```

The 5T types the five lines following the buffer pointer location. Note that the buffer pointer is currently at the beginning of the fourth line of the edit buffer.

No lines are printed by the OT command. If the buffer pointer is inside a line, the OT command will print the characters from the beginning of the line to the buffer pointer.

The -10T command prints the lines from the current buffer pointer location to the beginning of the edit buffer since there are less than 10 lines from the buffer pointer to the beginning of the edit buffer.

The command T prints the single line following the buffer pointer since the Editor assumes a one preceeds the T.

2-7.2 BUFFER POINTER OPERATIONS

Buffer pointer operations are used to manipulate the position of the edit buffer pointer.

2-7.2.1 B--BEGINNING

FORMAT: B

DESCRIPTION: The Beginning command moves the edit buffer pointer to the beginning of the edit buffer.

EXAMPLE:

```
@T$$
40  OPT S SELECT PRINTING SYMBOLS

@B$$

@T$$
10 NAM PGM

@
```

The line printed by the first T command is the fourth line in the edit buffer indicating that the buffer pointer is at the beginning of the fourth line.

The B command moves the buffer pointer to the beginning of the edit buffer.
The final T command causes the first line to be printed out confirming that the buffer pointer is at the beginning of the edit buffer.

2-7.2.2 Z--END OF BUFFER

FORMAT: Z

DESCRIPTION: The End of Buffer command moves the edit buffer pointer to the end of the edit buffer.

EXAMPLE:

```
    @T$$
    10 NAM PGM

    @Z$$

    @-1T$$
    330 MON

    @
```

Initially the buffer pointer is at the beginning of the edit buffer, as the first T command indicates by printing the first line.
The Z command moves the buffer pointer to the end of the edit buffer.
The -1T command prints the line immediately preceeding the buffer pointer location, in this case the last line of the edit buffer.

2-7.2.3 MOVE CHARACTER POINTER

FORMAT: nM

n is a decimal integer in the range -254 < n <255. If omitted, the value is assumed to be 1.

DESCRIPTION: The Move Character Pointer command moves the edit buffer pointer according to the number of characters specified by n. If n is positive, the pointer is moved forward n characters. If negative, the pointer is moved back n characters. If fewer than n characters are present between the initial buffer pointer and the end (beginning) of the edit buffer, the pointer is moved to the end (beginning) of the buffer.

EXAMPLE:

```
@4T$$
10 NAM PGM
20 OTP M MEMORY FILE OPTION
30 OPT O OUTPUT OBJECT TAPES
40 OPT S SELECT PRINTING SYMBOLS

@3M$$

@2T$$
NAM PGM
20 OTP M MEMORY FILE OPTION

@-1T$$
10
@8M$$

@T$$
20 OTP M MEMORY FILE OPTION

@-1M$$

@T$$

@
```

The 4T command prints the first 4 lines of the edit buffer indicating that the buffer pointer is at the beginning of the edit buffer.
The 3M command moves the buffer pointer forward 3 characters.
The 2T command demonstrates this by printing the current line, beginning at the buffer pointer location, and the next line.
The -1T command prints the previous line, which in this case is the information from the beginning of the edit buffer to the current buffer pointer location.

The buffer pointer is moved forward eight characters by the 8M command. Now the buffer pointer is at the beginning of the second line in the edit buffer as the T command demonstrates.
The -1M command moves the buffer pointer back one character to just before the Carriage Return at the end of the first line of the edit buffer. Now the T command prints only the Carriage Return, which is the remainder of the line it is in, and a Line Feed.

2-7.2.4 L--LINE

FORMAT: nL

n is a decimal integer in the range -254 < n < 255. If omitted, the value is assumed to be 1.

DESCRIPTION: The Line command moves the edit buffer pointer according to the number of lines specified by n. If n is positive, the pointer is moved forward n lines. If negative, the pointer is moved backward n lines. A value of 0 causes the pointer to be moved to the beginning of the current line. If fewer than n lines are present between the initial buffer pointer and the end (beginning) of the edit buffer, the pointer is moved to the end (beginning) of the buffer.

NOTE

The 680 Editor considers a line to be a sequence of characters delimited by Carriage Returns.

EXAMPLE:

```
@4T$$
10 NAM PGM
20 OTP M MEMORY FILE OPTION
30 OPT O OUTPUT OBJECT TAPES
40 OPT S SELECT PRINTING SYMBOLS

@3L$$

@T$$
40  OPT S SELECT PRINTING SYMBOLS
```

```
@-2L$$

@T$$
20 OTP M MEMORY FILE OPTION

@8M$$

@T$$
M MEMORY FILE OPTION

@0L$$

@T$$
20 OTP M MEMORY FILE OPTION

@
```

The 4T command prints the first four lines of the edit buffer indicating that the buffer pointer is at the beginning of the edit buffer.
The 3L command moves the buffer pointer forward three lines.
The T command then prints the fourth line of the edit buffer.
The -2L command moves the buffer pointer back two lines.
Now, the T command prints the second line of the edit buffer.
The 8M command moves the buffer pointer forward eight characters.
This is confirmed by the T command which prints only a portion of the second edit buffer line.
The command 0L causes the buffer pointer to move to the beginning of the line that it is currently in.
The final T command illustrates that the buffer pointer did move to the beginning of the second line.

2-7.2.5 S--SEARCH

FORMAT: Sstring

The "string" argument is a string of 16 characters or less, made up of any ASCII characters, except ESC and BREAK.

DESCRIPTION: The Search command causes a search of the edit buffer for the first occurrence of the specified string. The search begins at the buffer location specified by the current position of the buffer pointer. The search may be terminated in two ways:

(1) A match with the specified string is found. In this case, the buffer pointer is positioned immediately after the last character of the matched string.

(2) The search reaches the end of the edit buffer. In this case, a message

CAN'T FIND "string"

is printed. When no match is found, the buffer pointer remains in its initial position.

EXAMPLE:

```
@B$$

@S40$$

@T$$
  OPT S SELECT PRINTING SYMBOLS

@SOTP$$

CAN'T FIND "OTP"

@B$$

@SOTP$$

@OL$$
```

```
@T$$
20  OTP M MEMORY FILE OPTION

@
```

The B command sets the buffer pointer at the beginning of the edit buffer.
The S40 command searches for 40.
Since the Editor came back with @, it found 40 and positioned the buffer pointer immediately after it. The T command demonstrates this by printing the portion of the line numbered 40 which follows the 40. Note that if line numbers are included in the program being edited, the Search command can be used to easily move the buffer pointer to any given line by searching for the appropriate line number.
The SOTP command searches for OTP.
The Editor printed CAN'T FIND "OTP" indicating that OTP does not occur between the current buffer pointer location and the end of the edit buffer.
Now OTP is searched for again by re-entering the command SOTP.
This time the Editor prints only @, indicating that it found OTP. The OL command moves the buffer pointer to the beginning of the line containing OTP.
The T command prints the line containing OTP.

2-7.3 INSERT/DELETE/CHANGE OPERATIONS

These operations permit the insertion of text into the edit buffer, deletion of text in the buffer, or replacement of an existing character string with another string.

2-7.3.1 I--INSERT

FORMAT: Itext

The "text" argument may include any ASCII characters except ESC and CANCEL.

DESCRIPTION: The Insert command is used to insert lines or characters of text into the edit buffer. Text is inserted at the location specified by the current buffer pointer. Following the Insert operation, the pointer is positioned after the last character of inserted text. The ASCII characters Null, Rubout, Linefeed, Backspace, Readeron, Punchon, Readeroff, and Punchoff are deleted from the inserted text by the Editor.

EXAMPLE:

```
@B$$

@2T$$
10 NAM PGM
20 OTP M MEMORY FILE OPTION

@2M$$

@I $$

@L$$

@I15  * REVISION 1
$$

@B$$

@3T$$
10 NAM PGM
15 * REVISION 1
20 OTP M MEMORY FILE OPTION

@
```

The B command sets the buffer pointer at the beginning of the edit buffer.

The 2T command prints the first two lines of the edit buffer.
The 2M command moves the buffer pointer to immediately past the 0 to 10.
The I (Space) command inserts a space at the current location of the buffer pointer.
The L command moves the buffer pointer to the beginning of the next line so that a line can be inserted between the lines numbered 10 and 20.
The 115 *REVISION 1 (Carriage Return) command then inserts the line numbered 15 between the lines numbered 10 and 20.
The B command sets the buffer pointer at the beginning of the edit buffer.
The 3T command then prints the first three lines of the edit buffer and confirms that the information was inserted.

2-7.3.2 D--DELETE CHARACTERS

FORMAT: nD

n is a decimal integer in the range -254 < n < 255. If omitted, the value is assumed to be 1.

DESCRIPTION: The Delete Characters command deletes n characters from the edit buffer, beginning at the current position of the edit buffer pointer. If n is positive, n characters following the current pointer position are deleted. If negative, n characters preceding the current pointer position are deleted. If there are less than n characters between the edit buffer pointer and the end (beginning) of the edit buffer, then the characters will be deleted and the buffer pointer will point to the end (beginning) of the edit buffer.

EXAMPLE:

```
    @B$$

    @4T$$
    10 NAM PGM
    15  * REVISION 1
    20 OTP M MEMORY FILE OPTION
    30 OPT O OUTPUT OBJECT TAPES
```

```
@S15$$

@D$$

@STAPES$$

@-1D$$

@-4T$$
10 NAM PGM
15 * REVISION 1
20 OTP M MEMORY FILE OPTION
30 OPT O OUTPUT OBJECT TAPE
@
```

The B command sets the buffer pointer at the beginning of the edit buffer.
The 4T command prints the first four lines of the edit buffer.
The S15 command searches for 15 and locates the buffer pointer immediately after it.
The D command deletes the character after the buffer pointer, in this case a space.
The STAPES command searches for TAPES and positions the buffer pointer immediately after it.
The -1D command deletes the character before the buffer pointer, which is the S of TAPES.
The -4T command prints the four lines preceeding the buffer pointer and confirms that the changes were made.

2-7.3.3 K--KILL (DELETE) LINES

FORMAT: nK

n is a decimal integer in the range -254 < n < 255. If omitted, the value is assumed to be 1.

DESCRIPTION: The Kill Lines command is similar to the Delete Characters command, except that n specifies a number of lines to be deleted from the edit buffer, rather than a number of characters. If n is positive, n lines following the current pointer position are deleted. If negative, n lines preceding the current position are deleted. If fewer than n lines remain between the current pointer position and the end (beginning) of the edit buffer, then the lines are deleted and the buffer pointer will point to the end (beginning) of the edit buffer.

If n is zero, the characters between the buffer pointer and the immediately preceeding Carriage Return will be deleted.

EXAMPLES:

```
@B$$

@7T$$
10 NAM PGM
15 * REVISION 1
20 OTP M MEMORY FILE OPTION
30 OPT O OUTPUT OBJECT TAPE
40 OPT S SELECT PRINTING SYMBOLS
50 ORG 8192
60 LDA B ADDR

@S60$$

@0L$$

@K$$

@B$$

@7T$$
10 NAM PGM
15 * REVISION 1
20 OTP M MEMORY FILE OPTION
30 OPT O OUTPUT OBJECT TAPE
40 OPT S SELECT PRINTING SYMBOLS
50 ORG 8192
70 COUNT EQU @8  @ INDICATES OCTAL

@
```

The B command sets the buffer pointer to the beginning of the edit buffer.
The 7T command prints the first seven lines of the edit buffer.
The S60 command searches for 60 and sets the buffer pointer immediately after it.
The 0L command moves the buffer pointer to the beginning of the line numbered 60.
The K command deletes the line that the buffer pointer is currently at. If the buffer pointer were somewhere else besides the beginning of a line, the K command would delete the characters from the buffer pointer through the following Carriage Return.
40. The B command sets the buffer pointer to the beginning of the edit buffer.
The 7T command prints the first seven lines of the edit buffer confirming that the line numbered 60 was deleted.

2-7.3.4 C--CHANGE

FORMAT: Cstring1$string2
"string 1" and "string 2" are strings of 16 ASCII characters or less. These strings may include any ASCII characters except ESC and BREAK. The two strings need not be of the same length.
DESCRIPTION: The Change command searches the edit buffer from the current buffer pointer position. When the first occurrence of "string 1" is found, those characters are changed to "string 2". The buffer pointer will be moved to the end of "string 2".
If "string 1" cannot be found, the message

CAN'T FIND "string 1"

is printed, and the position of the buffer pointer is unchanged.

EXAMPLE:

```
    @B$$


    @5T$$
    10 NAM PGM
    15 * REVISION 1
    20 OTP M MEMORY FILE OPTION
    30 OPT O OUTPUT OBJECT TAPE
```

```
40 OPT S SELECT PRINTING SYMBOLS

@CSYMBOLS$OF SYMBOLS$$

@COTP$OPT$$

CAN'T FIND "OTP"

@B$$

@COTP$OPT$$

@B$$

@5T$$
10 NAM PGM
15 * REVISION 1
20 OPT M MEMORY FILE OPTION
30 OPT O OUTPUT OBJECT TAPE
40 OPT S SELECT PRINTING OF SYMBOLS

@
```

The B command sets the buffer pointer at the beginning of the edit buffer.
The 5T command then prints the first five lines of the edit buffer.
The CSYMBOLS$OF SYMBOLS command searches for SYMBOLS and substitutes for it OF SYMBOLS. The COTP$OPT command searches for OTP. However, the Editor prints CAN'T FIND "OTP" indicating that OTP does not occur between the current buffer pointer location and the end of the edit buffer.
The B command moves the buffer pointer to the beginning of the edit buffer so that the complete edit buffer can be searched for OTP.
This time the command COTP$OPT locates OTP and substitutes OPT for it.
The B command sets the buffer pointer at the beginning of the edit buffer.
The 5T command prints the first five lines of the edit buffer and confirms that the changes were made.

2-7.4 EXITING THE EDITOR

2-7.4.1 X--EXIT

FORMAT: X

DESCRIPTION: The EXIT command causes control to be returned to the 680 PROM Monitor.

2-7.4.2 G--GO TO ASSEMBLER

FORMAT: G

DESCRIPTION: The GO TO ASSEMBLER command transfers control to the 680 Assembler. If the Assembler is not in memory, the message

ASSEMBLER NOT IN MEMORY

is printed on the System Console Device and control returns to the 680 PROM Monitor.

2-7.5 EDITOR COMMAND CHAINING

The Altair 680 Editor can accept sequences of edit commands and associated arguments, terminated by two ESC characters. Commands with arguments which follow them must be separated from subsequent commands with a single ESC character.

Two ESC characters mark the end of a command string and cause the Editor to begin execution. Commands in a string are executed from left-to-right, in the order in which they were entered. All commands pre-ceeding an illegal command in the command chain are executed.

EDITOR COMMAND CHAINING EXAMPLES

The following example assumes that the information contained on the tape in the Append Example is the only contents of the edit buffer.

```
@B8T$$
10 NAM PGM
20 OTP M MEMORY FILE OPTION
30 OPT O OUTPUT OBJECT TAPES
40 OPT S SELECT PRINTING SYMBOLS
50 ORG 8192
60 LDA B ADDR
70 COUNT EQU @8  @ INDICATES OCTAL
```

```
80 START LDS *STACK INZ STACK POINTER

@2MI $OLTCS$$
10 NAM PGM

@S60$OLKS@$DI993$OLTCOTP$OPT$$
70 COUNT EQU @3  @ INDICATES OCTAL

CAN'T FIND "OTP"

@BTSOTP$OPT

@BCOTP$OPT$BLI15 * REVISION 1
$SSYMBOLS$ 7MIOF $$

@B8TE$$
10 NAM PGM
15 * REVISION 1
20 OPT M MEMORY FILE OPTION
30 OPT O OUTPUT OBJECT TAPE
40 OPT S SELECT PRINTING OF SYMBOLS
50 ORG 8192
70 COUNT EQU @3  @ INDICATES OCTAL
80 START LDS *STACK INZ STACK POINTER

@
```

The B8T command chain sets the buffer pointer at the beginning of the edit buffer and then prints the first eight lines of the buffer.

The next command chain, 2MI $OLTCS$$, does the following:

1. Moves the buffer pointer forward two characters (2M).
2. Inserts a space (I $).
3. Moves the buffer pointer to the beginning of the current line (OL).
4. Prints the line (T).
5. Moves the buffer pointer to just after the next S in the buffer, which is in TAPES, and then deletes the S (CS$$). Note that the C command can be used to delete character by not including a string 2 in the command. However, when used in this manner, the C command

can only occur as an individual command or at the end of a command chain since two ESC characters must occur together to the deletion of string 2.

The command chain

```
S60$0LKS@$DI993$0LTCOTP$OPT$$
```

performs the following actions:

1. Searches for 60 and moves the buffer pointer to just after it (S60$).
2. Moves the buffer pointer to the beginning of the line it is in (0L).
3. Deletes the line (K).
4. Searches for @ and moves the buffer pointer to just after it (S@$).
5. Deletes the next character, which is 8 (D).
6. Inserts a 3 (I9(backspace)3$). Note that a 9 was erroneously entered and was deleted using a backspace (Control H) character. The backspace character may be used to delete as many previous characters in a command as required. The Editor prints the character deleted by each backspace.
7. Moves the buffer pointer to the beginning of the line the buffer pointer is currently in (0L).
8. Types the line the buffer pointer is currently at (T).
9. Searches for OTP (COTP$OPT$$). But the Editor does not find it between the current buffer pointer location and the end of the buffer, as it indicates by printing CAN'T FIND "OTP".

The command chain BTSOTP$OPT was not executed since a Control X character terminated the command chain. The Control X character deletes all commands up to the last prompt and prints another prompt.

The command chain BCOTP$OPT$BLI15 * REVISION 1(CR)$SSYMBOLS$-7MIOF $$ does the following:

1. Moves the buffer pointer to the beginning of the edit buffer (B).
2. Changes OTP to OPT and moves the buffer pointer to just after OPT (COTP$OPT$).
3. Moves the buffer pointer back to the beginning of the edit buffer (B).
4. Moves the buffer pointer to the beginning of the next line (L).

5. Inserts the line 15 * REVISION 1 at the current buffer pointer location (I15 * REVISION 1 (CR)$). Note that to insert a line the buffer pointer is moved to the beginning of the line that is to follow the inserted line. Then the line is inserted using the I command. A Carriage Return should be the last character of the inserted line.
6. Searches for SYMBOLS and moves the buffer pointer to just after it (SSYMBOLS$).
7. Moves the buffer pointer back seven characters (-7M) which puts it at the beginning of SYMBOLS.
8. Inserts OF(space) at the current buffer pointer location (OF $$).

The following actions are performed by the command chain B8TE:

1. Moves the buffer pointer to the beginning of the edit buffer (B).
2. Prints the eight lines following the current buffer pointer location (8T).
3. Ends editing on the contents of the edit buffer by punching a tape of the buffer contents and any remaining tape in the System Reader Device (E).

2-8 EDITOR MESSAGES

Table 2-2 lists and identifies the Editor messages.

TABLE 2-1 EDITOR COMMAND SUMMARY

COMMAND	DESCRIPTION
A	Append. Appends input text from the System Reader Device to the edit buffer.
B	Beginning. Moves the edit buffer pointer to the beginning of the edit buffer.
Cstring1$ string2	Change. Replaces the first occurrence of "string 1" with "string 2".
nD	Delete. Deletes n characters from the edit buffer.
E	End. Terminates an edit operation by writing the contents of the edit buffer to the output tape.
F	Tape Leader/Trailer. Writes 50 NULL characters to the System Punch Device.
G	Go to Assembler. Passes control to the Altair 680 Assembler.
Istring	Insert. Inserts characters or lines of text into the edit buffer.
nK	Kill lines. Deletes n lines from the edit buffer.
nL	Line. Moves the edit buffer point n lines.
nM	Move character pointer. Moves the edit buffer pointer n characters.
nP	Punch. Punches n lines from the edit buffer to the System Punch Device.
Sstring	Search. Searches the edit buffer for the first occurrence of "string".
nT	Type. Types n lines from the edit buffer to the System Console Device.
X	Exit. Returns control to the 680 PROM Monitor.
Z	End of edit buffer. Moves the edit buffer pointer to the end of the edit buffer.
Control A	Causes the last character entered in the command mode to be typed on the System Console Device and deleted from the command.
Control X	Causes all commands following the last prompt to be deleted and another prompt to be typed.

TABLE 2-2. ALTAIR 680 EDITOR MESSAGES

MESSAGE	
ALTAIR 680 EDITOR N.N.	Printed upon initiation of Editor. Revision is specified by n.n.
@	Prompt. Editor is waiting for a command.
" " ????	Illegal command.
CAN'T FIND "string"	Editor cannot find the string specified by Search or Change command.
BELL	The Editor rings the bell in the System Console Device when the user attempts to enter further commands into a full command buffer. The user must delete (backspace) two characters in order to terminate the command with two ESC characters.
ASSEMBLER NOT IN MEMORY	Printed when the G command is issued and the 680 Assembler is not in memory.

CHAPTER 3 ALTAIR 680 ASSEMBLER

3-1. INTRODUCTION

The Altair 680 Assembler is used to translate M6800 MPU source programs written in assembly language mnemonics into machine executable object code. The format of the 680 assembly code source language is fully described in Chapter III of the Altair 680 Programming Manual.

3-2. ASSEMBLER INPUT

Commands to direct the operation of the 680 Assembler are input from the System Console Device. Source language programs may be input to the Assembler from the System Reader Device or read directly from an area in memory designated as the Text Buffer.

3-3. ASSEMBLER OUTPUT

The Assembler produces output in three forms:

1) An assembly listing
2) An object tape
3) A machine file (object program image in memory)

The assembly listing includes both a formatted output of the source program and a listing of the generated machine instructions. This listing is produced on the System Printer Device.

The object output is optionally written to the System Punch Device. Object tapes are punched in the format required for loading via the 680 PROM Monitor.

The memory file (OPT M) feature of the Assembler permits the object code to be loaded directly into memory during assembly. This feature facilitates execution of a program immediately after assembly, eliminating the need to punch and load an object tape.

3-4. LOADING THE ASSEMBLER

The Assembler is supplied on a tape labelled ALTAIR 680 ASSEMBLER/EDITOR which contains both the Assembler and Text Editor. This tape is loaded into memory using the PROM Monitor's L command. (See the 680 System Monitor Manual.)

3-5. ASSEMBLER INITIATION

If space for a user program is to be reserved in memory, then prior to starting the Assembler, the PROM Monitor's M and N commands should be used to deposit the address of the last memory location to be used by the Assembler into ENDSYS (locations D4 and D5). If this is not done, the Assembler will automatically determine the amount of contiguous RAM available and use all of it.

To start the Assembler, use the Monitor's J command and start execution at Ø1ØE. The Assembler will print the message

ALTAIR 680 ASSEMBLER X.X

where X.X is the version number of the Assembler.

If the Editor is in memory at the time the Assembler is started, the Assembler will ask the user if the Editor should be overwritten. Type Y (YES) or N (NO) followed by a Carriage Return to indicate whether the Editor should be overwritten.

3-6. ASSEMBLER OPERATION

The Altair 680 Assembler is a two-pass assembler. That is, the Assembler must read a source program twice--once to build a symbol table, and a second time to produce the assembled output. In response to the Assembler prompt message

ENTER PASS

select the appropriate response from the list of responses below. All responses must be terminated by a Carriage Return.

ØR - The ØR response reads a source tape into the Text Buffer. Reading of the tape terminates when a control Z character is encountered. Once the source file is stored in the Text Buffer, the Assembler must still execute pass 1 and pass 2. However, since the time required for the Assembler to read the source from memory is negligible, total assembly time can be reduced by as much as 50%. This method of assembly must be used whenever a source tape is assembled without the use of a controllable paper tape reader.

NOTE

If the Editor is also in memory, a source file read into the Text Buffer using the ØR pass will be available for editing when the Editor is re-entered.

ØC - The ØC response clears the Text Buffer. In order to assemble a program other than the program currently stored in the Text Buffer, it is necessary to first clear the Text Buffer.

1P - The 1P pass reads the source file (either from the System Reader Device or the Text Buffer) and produces a table of symbols which appear in the program and their corresponding numeric values. This table is used during pass 2 to evaluate the operand field of instructions which reference these symbols. Program syntax is also checked on pass 1, and errors are listed.

1S - The 1S pass is identical to the 1P pass with the exception that it does not clear the symbol table prior to reading the source file. This is useful in the assembly of multiple source tapes, as it permits all symbols to be known to each assembly.

2L - The 2L pass rereads the source file (either from the System Ready Device or the Text Buffer) and uses information in the symbol table to produce an assembly listing. Error messages are listed, and a machine file is created if the OPT M directive has been specified.

2T - The 2T pass is identical to the 2L pass except that it produces an object tape rather than an assembly listing. Error messages are listed, and a machine file is created if the OPT M directive has been specified.

E - The E response causes control to be transferred to the 680 Editor if it is in memory. If the Editor is not in memory, an error message will be printed and control will return to the PROM Monitor.

X - The X response causes control to be transferred to the 680 PROM Monitor.

3-7. USE OF THE MEMORY FILE OPTION

The memory file option allows the object code generated to be loaded directly into memory during pass two. The memory file option is specified by including an OPT M directive in the source program.

In order to use the memory file option, the address of the last memory byte to be used by the Assembler must be deposited into ENDSYS, locations D4-D5 (see Chapter 4, Procedure 1, Steps 2 and 3). This permits the Assembler to load the object code into memory locations above the address specified in ENDSYS. Any attempt to assemble a program that loads into locations less than or equal to the address specified in ENDSYS causes an error to be printed and the object code will not be loaded. Attempts to assemble into non-existent or faulty memory are also flagged as errors during the assembly process. When using the OPT M feature as described in Chapter 4, it is possible to have the program take advantage of the direct addressing mode by using locations 0000 - 0063 for temporary storage. The rest of page 0 is reserved for system use.

3-8. THE OPT DIRECTIVE

The OPT directive is used to control the assembly output. Multiple comma-separated options may be specified with a single statement.

OPT O (object tape)	The assembler will generate object tape (selected by default).
OPT NOO	No object tape.
OPT M (memory file)	The assembler will write machine code to memory.
OPT NOM	No memory (selected by default).
OPT S (print symbols)	The assembler will print the symbols at the end of Pass 2.
OPT NOS	No printing of symbols (selected by default).
OPT NOL (no listing)	The assembler will not print a listing of the assembled data.
OPT L	The listing of assembled data will be printed (selected by default).
OPT NOP (no page)	The assembler will inhibit format paging of the assembly listing (selected by default).
OPT P	The listing will be paged.
OPT NOG (no generate)	Causes only 1 line of data to be listed from the assembler directions FCC, FCB, FDB.
OPT G	All data generated by the FCC, FCB and FDB directions will be printed (selected by default).

3-9. ASSEMBLER ERROR MESSAGES

The following is a numerical list of Assembler error messages and their meanings.

ØØ1 - No END statement in program.

ØØ2 - Editor not in memory. An attempt was made to transfer control to the Editor while it was not in memory.

ØØ3 - Undefined opcode. The symbol in the opcode field is not a valid opcode mnemonic or directive.

ØØ4 - Text Buffer full. While reading a source file into the Text Buffer, the Text Buffer became full.

ØØ5 - Label error. The statement label field is not terminated.

ØØ6 - NAM directive error. The NAM directive is not the first source statement, it is missing, or it occurs more than once in the same source program.

ØØ7 - Label or opcode error. The label or opcode symbol does not begin with an alphabetic character.

ØØ8 - Syntax error. The program statement is syntactically incorrect.

ØØ9 - Byte overflow. An expression converted to a value greater than 255 (decimal).

Ø1Ø - OPT directive error. The structure of the OPT directive is syntactically incorrect or the option is undefined.

Ø11 - Branch error. The branch count is beyond the relative byte's range. The allowable range is:

$(*+2) - 128 < D < (*+2) + 127$

where: * = address of the first byte of the branch instruction.
D = address of the destination of the branch instruction.

Ø12 - Illegal addressing mode. The specified addressing mode is not allowed with the specified opcode.

Ø13 - Directive operand error. The directive's operand field is in error.

Ø14 - Redefined label. The statement label was previously defined. The first value is retained.

Ø15 - Redefined symbol. The symbol has been previously defined. The first value is retained.

Ø16 - Undefined symbol. The symbol does not appear in a label field.

Ø17 - Symbol table overflow. The symbol table has overflowed. The new symbol was not stored and all references to it will be flagged as errors.

Ø18 - Memory file error. An attempt was made to store object code below the address specified in ENDSYS. The object code was not loaded.

Ø19 - Faulty or non-existent memory. The memory file option attempted to store object code into faulty or non-existent RAM.

ADDENDA TO CHAPTERS 2 AND 3

The number of NULLs transmitted after each CRLF (Carriage Return/ Line Feed) is initially set to Ø. This can be altered by depositing the desired number of NULLs into location CE. For example, to set the number of NULLs to 3, the Monitor command

```
.M ØØCE ØØ Ø3
```

would be used.

CHAPTER 4. PROGRAM DEVELOPMENT PROCEDURES

4-1. INTRODUCTION

This chapter describes two procedures for program development using the 680 Editor and Assembler.

The first procedure outlined requires the Editor and Assembler to be co-resident in memory and uses the Edit Buffer to transfer the source program from the Editor to the Assembler. Since considerable time is saved by eliminating the creation and reading of external files, this procedure should be used whenever the amount of available memory permits.

The second procedure involves loading the Editor and Assembler independently of each other and using the tape that is generated by the Editor to transfer the source program to the Assembler. Although this procedure requires more I/O time, it uses less memory, thus permitting development of larger programs.

4-2. PROCEDURE 1

1) Use the PROM Monitor's L command to load the tape marked Altair 680 ASSEMBLER/EDITOR into memory.

2) Estimate the amount of memory the assembled program will require. Subtract that amount from the address of the highest memory location. The result will be the address of the last memory location to be used by the Assembler and Editor.

3) Use the PROM Monitor's M and N commands to deposit the address determined in Step 2 into ENDSYS (locations D4 and D5). This allows the rest of the memory to be used for the program under development. (In the example, ENDSYS is set to 3FFF hexadecimal which is 16K-1.) The sample program assembles into memory starting at location 4ØØØ.

4) Start the Editor by using the PROM Monitor's J command to begin execution at location Ø1Ø7.

5) Using the Editor commands outlined in Chapter 2, enter the program on the System Console Device and edit as necessary.

6) Use the Editor's G command to transfer control to the Assembler.

7) Respond N (No) to the Assembler's question, "OVERWRITE EDITOR?".

8) Type 1P when the Assembler prompts, "ENTER PASS". This causes the Assembler to execute Pass 1 of the assembly by reading the program from the Edit Buffer.

9) If no errors are indicated by the Assembler, proceed to step 13. Otherwise, follow steps 10 through 12. (In the example, two errors were indicated.)

10) Type E in response to the Assembler's prompt, "ENTER PASS" to re-enter the Editor.

11) Use the Editor to make the necessary corrections to the source program.

12) Go back to step 6.

13) Type 2L when the Assembler prompts, "ENTER PASS". This causes an assembly listing to be produced which proves invaluable during program debugging. Since the machine file option was specified (OPT M), the object code is assembled into memory during the 2L pass.

14) If no assembly errors are indicated during the 2L pass, proceed to step 15. Otherwise, go back to step 1Ø.

15) Type X in response to the Assembler's prompt, "ENTER PASS". This causes control to be returned to the PROM Monitor.

16) Use the PROM Monitor's J command to begin execution at the start of the program.

17) Test and debug the program as necessary.

18) If the program performs properly, proceed to step 21. Otherwise, follow steps 19-20.

19) Use the Monitor's J command to re-enter the Editor at location Ø1ØA.

20) Go back to step 11.

21) At this point the program has been fully developed. It may be desirable to perform steps 22-26 through to create a source tape and an object tape for future use.

22) Use the Monitor's J command to re-enter the Editor at location Ø1ØA.

23) Use the Editor's F and E commands to punch a source tape of the program.

24) Use the Editor's G command to pass control to the Assembler.

25) Execute pass 1P of the Assembler.

26) Execute pass 2T of the Assembler to punch an object tape of the program.

```
.L
S10400F3FF08
S9
.M 00D4 00 3F
.N 00D5 00 FF
.J 0107
ALTAIR 680 EDITØR 1.0

ØI NAM STICKS
 ØPT P
 ØPT S
 ØPT NØG
 ØPT M
RESET EQU $FFFE PRØM MØNITØR RESET VECTØR
ØUTCH EQU $FF81 PRØM MØNITØR ØUTPUT CHAR
INCH EQU $FF00 PRØM MØNITØR INPUT CHAR
STACK EQU 30 STACK WILL BE ØN PAGE ZERØ
 ØRG $4000 SET LØC CØUNTER TØ 16K
START LDS #STACK INIT THE STACK PØINTER
 LDX #INTRØ INTRØDUCE MYSELF
 BSR PMESS
 LDA A #21 INIT PILE TØ 21 STICKS
HUMAN BSR EVAL PRINT #ØF STICKS
RETAD1 LDX #ASK ASK HØW MANY
 BSR PMESS
 JSR INCH GET RESPØNSE
 CMP B #'1 IS IT A 1?
 BEQ ØKRESP YES, RESPØNSE IS ØK
 CMP B #'2 NØ, HØW ABØUT A 2?
 BEQ ØKRESP YES, RESPØNSE IS ØK
 LDX #ERRMES NØ, SEND ERRØR MESSAGE
 BSR PMESS
 BRA RETAD1 ASK THEM AGAIN
ØKRESP SUB B #'0 SUBTRACT ASCII ZERØ
 PSH A SAVE # IN PILE
 SBA SUBTRACT WHAT THEY TØØK
 BPL ALLØK NØN NEGATIVE-ALL ØK
 PUL,A THERE AREN'T THAT MANY
 LDX #ERRM2 SEND AN ERRØR MESSAGE
 BSR PMESS
 BRA RETAD1 ASK THEM AGAIN
ALLØK INS TAKE GARBAGE ØFF STACK
 BSR EVAL PRINT AND CHECK
 PSH A SAVE # ØF STICKS
 LDA B #2 I'LL TAKE 2 STICKS IF
SUB3 SUB A #3 PILE CØNTAINS MULT ØF 3
 BEQ TAKEM ØTHERWISE I'LL TAKE 1
 BPL SUB3
 DEC B
TAKEM PUL A RESTØRE PILE
 SBA SUBTRACT WHAT I TØØK
 LDX #ITAKE INFØRM THE HUMAN AS TØ HØW
 ADD B #'0 MANY STICKS I TØØK
 STA B 7,X
 BSR PMESS PRINT MESSAGE
 BRA HUMAN GIVE HIM ANØTHER CHANCE
*
* PMESS PRINTS A CHAR STRING PØINTED TØ BY X
* STØPS WHEN IT FINDS CHAR WITH BIT 7 ØN
*
PMESS1 INX BUMP PØINTER
PMESS LDA B X GET CHAR ØF STRING
 JSR ØUTCH SEND IT TØ TERMINAL
 BPL PMESS1 CØNTINUE IF BIT 7 IS LØW
 RTS RETURN IF BIT 7 IS HIGH
*
* EVAL PRINTS THE NUMBER ØF STICKS
* REMAINING IN THE PILE AND DETERMINES
* IF THE GAME IS ØVER AND WHØ WØN
*
EVAL LDX #NUMSTK PØINT TØ #ØF STICKS MESSAGE
 LDA B #$FF CØNVERT # ØF STICKS TØ DECIMAL
 PSH A
SUB10 INC B AND PRINT IT
 SUB A #10
 BCC SUB10
 ADD B #'0 PUT THE # IN THE MESSAGE
 STA B 16,X
 ADD A #@72
 STA A 17,X
 PUL A
 BSR PMESS
 TST A IS PILE REDUCED TØ ZERØ?
 BEQ DØNE YES, GAME IS ØVER
 RTS NØ, KEEP ØN PLAYING
DØNETSX WHØ WØN?
 LDX X IF WE WØULD RETURN TØ
 CPX #RETAD1 RETAD1 THEN HE WØN
 BNE IDID
 LDX #HEWØN THE HUMAN WØN
 BRA PRINT
IDID LDX #IWØN
PRINT BSR PMESS
 LDX RESET GØ BACK TØ PRØM MØNITØR
 JMP X
*
* HERE ARE THE MESSAGES
*
INTRØ FCB @15,@12 CARRIAGE RETURN LINE FEED
 FCC /LET'S PLAY STICKS/
 FCB @15,@12
 FCC /WE HAVE A PILE ØF STICKS/
 FCB @15,@12
 FCC /WE TAKE TURNS REMØVING 1 ØR 2 STICKS/
 FCB @15,@12
 FCC /THE PERSØN (ØR CØMPUTER) WHØ TAKES THE LAST STICK LØSES/
 FCB @15,@212
```

```
NUMSTK FCB @15,@12
 FCC /THERE ARE NØW    STICKS/
 FCB @15,@212
ASK FCC /HØW MANY STICKS DØ YØU TAKE?/
 FCB @240
ITAKE FCC /I TAKE   STICK/
 FCB @323
HEWØN FCC /YØU WØN - YØU PRØBABLY CHEATED!!/
 FCB @15,@212
IWØN FCC /I WØN - AS USUAL!!!!!/
 FCB @15,@212
ERRMES FCB @15,@12
 FCC /YØU CAN ØNLY TAKE 1 ØR 2 STICKS!/
 FCB @15,@212
ERRM2 FCB @15,@12,7
 FCC /YØU CAN'T DØ THAT!!!/
 FCB @15,@12
 FCC /I CØMMAND YØU TØ TAKE THE 1 REMAINING STICK!!!/
 FCB @15,@212
 END
 SS
 @GSS


ALTAIR 680 ASSEMBLER 1.0
ØVERWRITE EDITØR? N
ENTER PASS 1P

****ERRØR  008
****ERRØR  012
00030 4029 00 0000  PUL,A THERE AREN'T THAT MANY

****ERRØR  003
00078 4070 00 FFFE DØNETSX WHØ WØN?



ENTER PASS E
ALTAIR 680 EDITØR 1.0

@BCPUL,ASPUL ASCDØNETSXSDØNE TSXSGSS
ALTAIR 680 ASSEMBLER 1.0
ØVERWRITE EDITØR? N
ENTER PASS 1P


ENTER PASS 2L
```

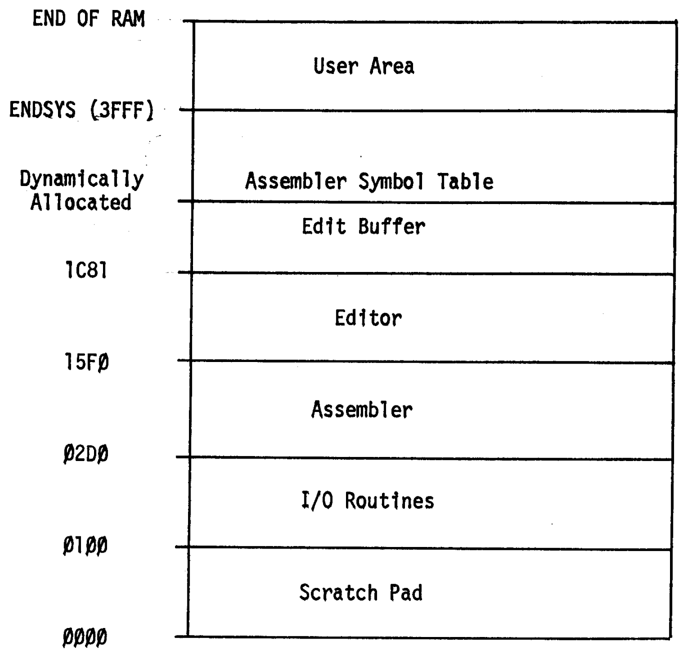
```
PAGE 001    STICKS

                          NAM     STICKS
                          ØPT     S
                          ØPT     P
                          ØPT     NØG
                          ØPT     M
      FFFE      RESET     EQU     $FFFE      PRØM MØNITØR RESET VECTØR
      FF81      ØUTCH     EQU     $FF81      PRØM MØNITØR ØUTPUT CHAR
      FF00      INCH      EQU     $FF00      PRØM MØNITØR INPUT CHAR
      001E      STACK     EQU     30         STACK WILL BE ØN PAGE ZERØ
4000                      ØRG     $4000      SET LØC CØUNTER TØ 16K
4000  8E 001E   START     LDS     #STACK     INIT THE STACK PØINTER
4003  CE 4085             LDX     #INTRØ     INTRØDUCE MYSELF
4006  8D 44               BSR     PMESS
4008  86 15               LDA A   #21        INIT PILE TØ 21 STICKS
400A  8D 48     HUMAN     BSR     EVAL       PRINT #ØF STICKS
400C  CE 412E   RETAD1    LDX     #ASK       ASK HØW MANY
400F  8D 3B               BSR     PMESS
4011  BD FF00             JSR     INCH       GET RESPØNSE
4014  C1 31               CMP B   #'1        IS IT A 1?
4016  27 0B               BEQ     ØKRESP     YES, RESPØNSE IS ØK
4018  C1 32               CMP B   #'2        NØ, HØW ABØUT A 2?
401A  27 07               BEQ     ØKRESP     YES, RESPØNSE IS ØK
401C  CE 4193             LDX     #ERRMES    NØ, SEND ERRØR MESSAGE
401F  8D 2B               BSR     PMESS
4021  20 E9               BRA     RETAD1     ASK THEM AGAIN
4023  C0 30     ØKRESP    SUB B   #'0        SUBTRACT ASCII ZERØ
4025  36                  PSH A              SAVE # IN PILE
4026  10                  SBA                SUBTRACT WHAT THEY TØØK
4027  2A 08               BPL     ALLØK      NØN NEGATIVE-ALL ØK
4029  32                  PUL A              THERE AREN'T THAT MANY
402A  CE 41B7             LDX     #ERRM2     SEND AN ERRØR MESSAGE
402D  8D 1D               BSR     PMESS
402F  20 DB               BRA     RETAD1     ASK THEM AGAIN
4031  31        ALLØK     INS                TAKE GARBAGE ØFF STACK
4032  8D 20               BSR     EVAL       PRINT AND CHECK
4034  36                  PSH A              SAVE # ØF STICKS
4035  C6 02               LDA B   #2         I'LL TAKE 2 STICKS IF
4037  80 03     SUB3      SUB A   #3         PILE CØNTAINS MULT ØF 3
4039  27 03               BEQ     TAKEM      ØTHERWISE I'LL TAKE 1
403B  2A FA               BPL     SUB3
403D  5A                  DEC B
403E  32        TAKEM     PUL A              RESTØRE PILE
403F  10                  SBA                SUBTRACT WHAT I TØØK
4040  CE 414B             LDX     #ITAKE     INFØRM THE HUMAN AS TØ HØW
4043  CB 30               ADD B   #'0        MANY STICKS I TØØK
4045  E7 07               STA B   7,X
4047  8D 03               BSR     PMESS      PRINT MESSAGE
4049  20 BF               BRA     HUMAN      GIVE HIM ANØTHER CHANCE
                *
                * PMESS PRINTS A CHAR STRING PØINTED TØ BY X
                * STØPS WHEN IT FINDS CHAR WITH BIT 7 ØN
                *
404B  08        PMESS1    INX                BUMP PØINTER
404C  E6 00     PMESS     LDA B   X          GET CHAR ØF STRING
```

```
404E BD FF81        JSR     @UTCH    SEND IT TØ TERMINAL
4051 2A F8          BPL     PMESS1   CØNTINUE IF BIT 7 IS LØW
4053 39             RTS              RETURN IF BIT 7 IS HIGH
                *
                * EVAL PRINTS THE NUMBER ØF STICKS
                * REMAINING IN THE PILE AND DETERMINES
                * IF THE GAME IS ØVER AND WHØ WØN
                *
4054 CE 4113    EVAL    LDX     #NUMSTK  PØINT TØ #ØF STICKS MESSAG
4057 C6 FF              LDA B   #$FF     CØNVERT # ØF STICKS TØ DEC
4059 36                 PSH A
405A 5C         SUB10   INC B            AND PRINT IT
405B 80 0A              SUB A   #10
405D 24 FB              BCC     SUB10
405F CB 30              ADD B   #'0      PUT THE # IN THE MESSAGE
4061 E7 10              STA B   16,X
4063 8B 3A              ADD A   #@72
4065 A7 11              STA A   17,X
4067 32                 PUL A
4068 8D E2              BSR     PMESS
406A 4D                 TST A            IS PILE REDUCED TØ ZERØ?
406B 27 01              BEQ     DØNE     YES, GAME IS ØVER
406D 39                 RTS              NØ, KEEP ØN PLAYING
406E 30         DØNE    TSX              WHØ WØN?
406F EE 00              LDX     X        IF WE WØULD RETURN TØ
4071 8C 400C            CPX     #RETAD1  RETAD1 THEN HE WØN
4074 26 05              BNE     IDID
4076 CE 415A            LDX     #HEWØN   THE HUMAN WØN
4079 20 03              BRA     PRINT
407B CE 417C    IDID    LDX     #IWØN
407E 8D CC      PRINT   BSR     PMESS
4080 FE FFFE            LDX     RESET    GØ BACK TØ PRØM MØNITØR
4083 6E 00              JMP     X
                *
                * HERE ARE THE MESSAGES
                *
4085 0D         INTRØ   FCB     @15,@12  CARRIAGE RETURN LINE FEED
4087 4C                 FCC     /LET'S PLAY STICKS/
4098 0D                 FCB     @15,@12
409A 57                 FCC     /WE HAVE A PILE ØF STICKS/
40B2 0D                 FCB     @15,@12
40B4 57                 FCC     /WE TAKE TURNS REMØVING 1 ØR 2 STI
40D8 0D                 FCB     @15,@12
40DA 54                 FCC     /THE PERSØN (ØR CØMPUTER) WHØ TAKE
4111 0D                 FCB     @15,@212
4113 0D         NUMSTK  FCB     @15,@12
4115 54                 FCC     /THERE ARE NØW    STICKS/
412C 0D                 FCB     @15,@212
412E 48         ASK     FCC     /HØW MANY STICKS DØ YØU TAKE?/
414A A0                 FCB     @240
414B 49         ITAKE   FCC     /I TAKE   STICK/
4159 D3                 FCB     @323
415A 59         HEWØN   FCC     /YØU WØN - YØU PRØBABLY CHEATED!!/
417A 0D                 FCB     @15,@212
```

```
00109 417C 49        IWØN   FCC     /I WØN - AS USUAL!!!!!/
00110 4191 0D               FCB     @15,@212
00111 4193 0D        ERRMES FCB     @15,@12
00112 4195 59               FCC     /YØU CAN ØNLY TAKE 1 ØR 2 STICKS!/
00113 41B5 0D               FCB     @15,@212
00114 41B7 0D        ERRM2  FCB     @15,@12,7
00115 41BA 59               FCC     /YØU CAN'T DØ THAT!!!/
00116 41CE 0D               FCB     @15,@12
00117 41D0 49               FCC     /I CØMMAND YØU TØ TAKE THE 1 REMAI
00118 41FE 0D               FCB     @15,@212
00119                       END

RESET   FFFE
ØUTCH   FF81
INCH    FF00
STACK   001E
START   4000
HUMAN   400A
RETAD1  400C
ØKRESP  4023
ALLØK   4031
SUB3    4037
TAKEM   403E
PMESS1  404B
PMESS   404C
EVAL    4054
SUB10   405A
DØNE    406E
IDID    407B
PRINT   407E
INTRØ   4085
NUMSTK  4113
ASK     412E
ITAKE   414B
HEWØN   415A
IWØN    417C
ERRMES  4193
ERRM2   41B7

TØTAL ERRØRS 00000
```

```
ENTER PASS X
.J 4000
LET'S PLAY STICKS
WE HAVE A PILE ØF STICKS
WE TAKE TURNS REMØVING 1 ØR 2 STICKS
THE PERSØN (ØR CØMPUTER) WHØ TAKES THE LAST STICK LØSES

THERE ARE NØW 21 STICKS
HØW MANY STICKS DØ YØU TAKE? 3
YØU CAN ØNLY TAKE 1 ØR 2 STICKS!
HØW MANY STICKS DØ YØU TAKE? 2
THERE ARE NØW 19 STICKS
I TAKE 1 STICKS
THERE ARE NØW 18 STICKS
HØW MANY STICKS DØ YØU TAKE? 1
THERE ARE NØW 17 STICKS
I TAKE 1 STICKS
THERE ARE NØW 16 STICKS
HØW MANY STICKS DØ YØU TAKE? 2
THERE ARE NØW 14 STICKS
I TAKE 1 STICKS
THERE ARE NØW 13 STICKS
HØW MANY STICKS DØ YØU TAKE? 2
THERE ARE NØW 11 STICKS
I TAKE 1 STICKS
THERE ARE NØW 10 STICKS
HØW MANY STICKS DØ YØU TAKE? 2
THERE ARE NØW 08 STICKS
I TAKE 1 STICKS
THERE ARE NØW 07 STICKS
HØW MANY STICKS DØ YØU TAKE? 2
THERE ARE NØW 05 STICKS
I TAKE 1 STICKS
THERE ARE NØW 04 STICKS
HØW MANY STICKS DØ YØU TAKE? 2
THERE ARE NØW 02 STICKS
I TAKE 1 STICKS
THERE ARE NØW 01 STICKS
HØW MANY STICKS DØ YØU TAKE? 2
YØU CAN'T DØ THAT!!!
I CØMMAND YØU TØ TAKE THE 1 REMAINING STICK!!!
HØW MANY STICKS DØ YØU TAKE? 1
THERE ARE NØW 00 STICKS
I WØN - AS USUAL!!!!!
.
```

(Steps 23 through 26 of Procedure 1 are not shown on the output.)



Memory Map for Procedure 1

4-3. PROCEDURE 2

NOTE

No output is shown for this procedure as it would be essentially identical to the output of Procedure 1.

1) Use the Monitor's L command to load the tape marked ALTAIR 680 EDITOR.

2) Use the Monitor's J command to begin execution of the Editor at Ø1Ø7.

3) Using the Editor commands outlined in Chapter 2, enter the program on the System Console Device and edit as necessary.

4) Use the Editor's F and E commands to punch a source tape of the program.

5) Use the Editor's X command to return to the Monitor.

6) Load the tape marked ALTAIR 680 ASSEMBLER/EDITOR.

7) Start execution of the Assembler at Ø1ØE.

8) Respond Y (Yes) to the question "OVERWRITE EDITOR?".

9) Place the source tape in the System Reader Device.

10) Type 1P when the Assembler prompts, "ENTER PASS".

11) If no errors are indicated, proceed to step 18. Otherwise, follow steps 12-17.

12) Type X in response to the prompt, "ENTER PASS". Control will be returned to the Monitor.

13) Reload the tape marked ALTAIR 680 EDITOR.

14) Start execution at Ø1Ø7.

15) Use the A (Append) command to read the source tape from the System Reader Device into the Edit Buffer.

16) Edit the program as necessary.

17) Go back to step 4.

18) Type 2L to the prompt, "ENTER PASS".

19) An assembly listing will be produced.

20) If errors are indicated, go back to step 12. Otherwise, proceed to step 21.

21) Type 2T in response to the prompt, "ENTER PASS".

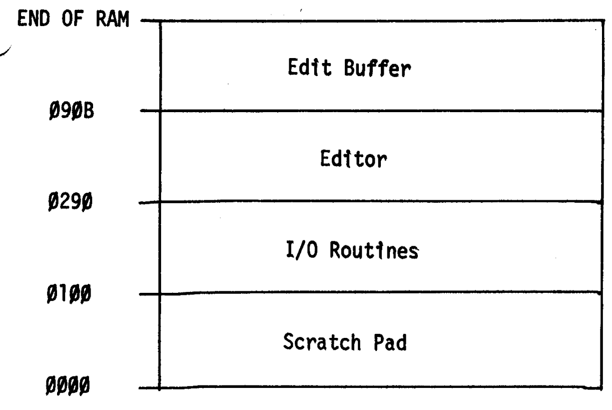
22) An object tape of the program will be punched on the System Punch Device.

23) Type X in response to the prompt, "ENTER PASS", to return to the Monitor.

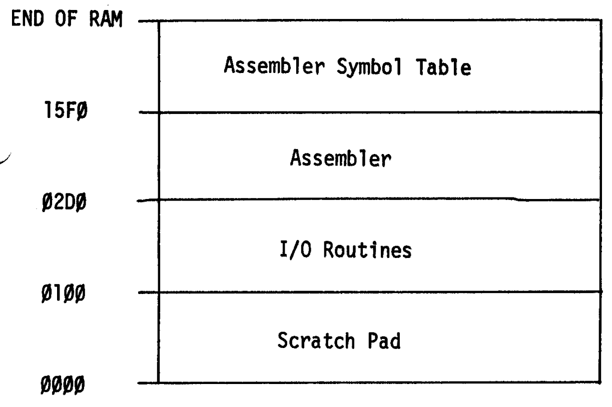
24) Use the Monitor's L command to load the object tape.

25) Test and debug the program.

26) If the program performs properly, the development procedure is complete. Otherwise, go back to step 13.



Editor



Assembler

Memory Maps for Procedure 2