6502
MACRO ASSEMBLER
AND
TEXT EDITOR
FOR PET, APPLE, SYM and OTHERS

<table>
<thead>
<tr>
<th>ASSEMBLE LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100 ;MOVE FROM TABLE1 TO TABLE2</td>
</tr>
<tr>
<td>0110 .BA $400</td>
</tr>
<tr>
<td>0400- AO 00 0120 START LDY #00</td>
</tr>
<tr>
<td>0402- 90 OB 04 0130 LOOP LDA TABLE1,Y</td>
</tr>
<tr>
<td>0405- 90 OB 05 0140 STA TABLE2,Y</td>
</tr>
<tr>
<td>0408- C8 0150 INY</td>
</tr>
<tr>
<td>0409- DE F7 0160 BNE LOOP</td>
</tr>
<tr>
<td>0165</td>
</tr>
<tr>
<td>0170 ;</td>
</tr>
<tr>
<td>040B- 0180 TABLE1 .DS 256 ;STORAGE</td>
</tr>
<tr>
<td>050B- 0190 TABLE2 .DS 256 ; *</td>
</tr>
<tr>
<td>0200 ;</td>
</tr>
<tr>
<td>0210 .EN</td>
</tr>
</tbody>
</table>

LABEL FILE \(/ = EXTERNAL\)

START= 0400 LOOP=0402 TABLE1=040B
TABLE2=050B

//0000,060B,060B

This ASSEMBLER and TEXT EDITOR was written in machine language-not BASIC
1. INTRODUCTION

This 6502 relocating Macro assembler (ASSM) and text editor (TED) resides simultaneously in approximately 8K bytes of memory. The ASSM/TED can be loaded into RAM or stored in ROM memory. Sufficient memory must be provided for not only the ASSM/TED but for a text file and label file (symbol table). Approximately 2K is sufficient memory for the text file for small programs or larger programs if assembled from tape. A good rule of thumb is one byte of memory for the label file for each byte of object code. If an executable object code file is to be stored in memory during assembly, sufficient memory must be provided for that also. On cold start entry (2000), the ASSM/TED will set the file boundaries as follows.

- Text file =
- Label file =
- Relocatable Object buffer =

See part 13

The label file and text file that this ASSM/TED generates is position independent and may be located practically anywhere in RAM memory. The object code file location is dependent on the beginning of assembly (.BA pseudo op) and the .MC pseudo op.

The ASSM/TED was designed such that records in the label file and text file are variable in length and directly dependent on the number of characters to be stored. This results in more efficient utilization of memory.

Some unique features of this ASSM/TED are:

- Macro and conditional assembly support.
- Labels up to 10 characters in length.
- Auto line numbering for ease of text entry.
- Creates both executable code in memory and relocatable object code on tape.
- Manuscript feature for composing letters and other text.
- Loading and storing of text on tape.
- Vectors for linkage to disc operating systems.
- Supports up to two tape decks, CRT and keyboard, and printer.
- String search and replace capability, plus other powerful editing commands.

Throughout this document, output generated by the ASSM/TED is underlined to distinguish from user input.
Initial entry to the ASSM/TED is at address 2000. If the break command (BREAK) is executed, one may return to the address following the break. Initial entry provides the following default parameters:

- Format - set
- Manuscript - clear
- Auto line numbering - 0 or clear
- Text file - clear
- Tape decks - off

The ASSM/TED is designed to operate with a record deck and a separate play deck and/or disc system. A single record/play deck may be used but one will not be able to create relocatable object files when assembling from tape.

This software has been extensively tested and is believed to be entirely reliable. It would be foolish to guarantee a program of this size and complexity to be free of errors. Therefore, we assume no responsibility for the failure of this software.

This ASSM/TED is protected by a copyright. This material may not be copied, reproduced or otherwise duplicated without written permission from the owner, Carl Moser. The purchaser may however make copies of this software on his storage medium (such as paper, disc, or magnetic tape) for his own personal use. The purchase of this software does not convey any license to manufacture, modify and/or copy this product. Providing copies of this software to friends or associates without authorization is a violation of Federal law.

2. TEXT EDITOR (TED) FEATURES

The TED occupies approximately one-half the total memory space of this software. The purpose of the TED is to setup and maintain the source file by interacting with the user via various commands.

When inputting to the TED, the user has the following options:

- Control H (hex 08) or
- RUBOUT (hex 7F) - Deletes previous character. More than one of these may be entered to delete a number of characters

- Control X (hex 18) - Deletes the entire line.

- Break - Halts outputting, and waits for input of appropriate control code (part II).
A. Commands

The TED provides 27 command functions. Each command mnemonic must begin immediately after the prompter (>). When entered, a command is not executed until a carriage return is given. Although a command mnemonic such as >PR may be several non-space characters in length, the ASSM/TED only considers the first two. For example, >PR, >PRI, >PRINT, and >PRETTY will be interpreted as the print command.

Some commands can be entered with various parameters. For example, >PRINT 10 200 will print out the text in the text file with line numbers between 10 and 200. One must separate the mnemonic and the parameters from one another by at least one space. - Do not use commas.

A description of each command follows:

> AUTO x

Automatic line numbering occurs when an x value not equal zero is entered. x specifies the increment to be added to each line number. Auto line numbering starts after one enters the first line. To prevent auto line numbering from reoccurring, enter > Au or > Au 0.

> GET Fx y

Get text file with data associated with file number x from tape or disc. The data will be loaded at line number y, or will be appended to end of the text file if the keyword APPEND is entered for y. Defaults are x=00 and y = line number 0.

Examples:

> GET F13 100
> GET APPEND

> PUT Fw x y

Put text file between lines x and y to tape or disc, and assign the recorded data file number w. If w is not entered, 00 will be assumed. If x and y are not entered, the entire text file is recorded. If the letter X is entered as the parameter such as

> PUT X and end of file mark is recorded.

> NUMBER x y

Renumber the text file starting at line x in text file and expanding by constant y. For example to renumber the entire text file by 10, enter > NU 0 10.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;DELETE x y</td>
<td>Delete entries in text file between line numbers x and y. If only x is entered, only that line is deleted.</td>
</tr>
<tr>
<td>&gt;OUTPUT Pw</td>
<td>Create a relocatable object file on tape deck 0 and assign file number w to the recorded data. If w is not entered 00 will be assumed. This command uses the 256 byte relocatable buffer that can be reallocated via the &gt;SET command.</td>
</tr>
<tr>
<td>&gt;HARD w x</td>
<td>Control output to hard copy output device (printer). Turn on outputting (w = SET) or turn off (w = CLEAR). The starting page number is x. This command is designed to leave a small margin at top and bottom, and provide a page number heading at the top of each page. It is designed to work with 66 line pages. An entry of &gt;HA PAGE results in the printer advancing to the top of the next page.</td>
</tr>
<tr>
<td>&gt;PRINT x y</td>
<td>Print the text file data between line number x and y on the CRT. If only x is entered, only that line is printed. If no x and y, the entire file is outputted.</td>
</tr>
<tr>
<td>&gt;ASSEMBLE w x</td>
<td>Clear the label file and then assemble source in the text file starting at line number x or 0 if x is not entered. If w=LIST then a listing will be generated. If w=NOLIST or LIST not entered then an errors only output will be generated.</td>
</tr>
<tr>
<td>&gt;RUN label</td>
<td>Run (execute) a previously assembled program. If a symbolic label is entered, the label file is searched for the starting address. The called program should contain an RTS instruction as the last executable instruction.</td>
</tr>
<tr>
<td>&gt;LABELS</td>
<td>Print out the label file.</td>
</tr>
</tbody>
</table>
PASS
Execute the second pass of assembly. Not required if source is
all in internal memory and the .CT pseudo op is not encountered.

FORMAT w
Format the text file (where w = SET) or clear the format feature
(where w = CLEAR). Format set tabulates the text file when
outputted. This lines up the various source statement fields.
This feature, set or clear, does not require extra memory.
Assembly output is dependent on the state of the format feature.

DUPLICATE Fw
Duplicate files from tape 1 to tape 0 until file w. This command
starts by reading the next file on tape 1 (or the disc input)
and if that file is file w or an end of file (EOF) mark then
it stops. If not, the file just read will be written to tape
ζ (or the disc output) and then tape 1 is read again. This
continues until file w or EOF is encountered.

COPY x y z
Copy lines y thru z in the text file to just after line number
x. The copied lines will all have line numbers equal x. At
completion, there will be two copies of this data - one at x
and the original at y.

MOVE x y z
Move lines y thru z in the text file to just after line number
x. The moved lines will all have line numbers equal x. The
original lines y thru z are deleted.

SET ts te ls le bs
If no parameters are given, the text file, label file, and
relocatable buffer boundaries (addresses indicating text file
start, end, label file start, end, and relocatable buffer start)
will be output on first line, then on the second line the output
consists of the present end of data in the text file followed
with the present end of data in the label file. This command
is commonly used to determine how much memory is remaining
in the text file. If you are inputting hex digits for these
addresses, proceed each with a ' $' character.
If parameters are entered, the first two are text file start (ts) and end (te) addresses, then the label file start (ls) and end (le) addresses, and finally the relocatable buffer start address (bs).

>USER

User defined command. The ASSM/TED will transfer control to location $0003. The user routine can reenter ASSM/TED via a warm start.

>ENTER filename

Enter a filename in the disc directory. This opens a disc output file. If no filename is entered, the result is a close operation. See parts 6 and 7 for details.

>LOOKUP filename

Look up a filename in the disc directory. This opens a disc input file. If no filename is entered, the result is a close operation. See parts 6 and 7 for details.

>FIND ts1t

Find string S1. See part 10B for details.

>MANUSCRIPT w

If w = SET, line numbers are not outputted when executing the >PR command. If w = CLEAR, line numbers are outputted when the >PR command is executed. Assembly output ignores the >MA command. If manuscript is to be generated with this ASSM/TED, manuscript should be set and format clear (>MA SET, >FO CLEAR). Since the TED considers a blank line a deletion, one must enter a non printable control character to trick the TED into inserting a blank line.

>ON n

Turn on tape deck n (where n is 0 (record), or 1 (play) deck). If an n is not entered, 0 is assumed.
8.

>OFF n
Turn off tape deck n (where n is 0 (record), or 1 (play) deck). If an n is not entered, 0 is assumed.

>CLEAR
Clear text file and turn off tape decks.

>BREAK
Break to monitor (executes BRK instruction). A return to the TED can be performed at the address immediately after the break instruction. (A control C operation does the same thing).

>R
Any entry beginning with one or more decimal digits is considered and entry/deletion of text. Details on this follows.

>EDIT tS1tS2t or EDIT n
See part 10A.

B. Entry/Deletion of Text

Source is entered in the text file by entering a line number (0-9999) followed by the text to be entered. The line number string can be one to n digits in length. If the string is greater than 4 digits in length, only the right-most 4 are considered. Text may be entered in any order but will be inserted in the text file in numerical order. This provides for assembling, printing, and recording in numerical order. Any entry consisting of a line number with no text or just spaces results in a deletion of any entry in the text file with the same number. If text is entered and a corresponding line number already exists in the text file, the text with the corresponding number is deleted and the entered text is inserted.

To delete the entire file, use the >CL command.

To delete a range of lines, use the >DE command. To edit an existing line or lines having similar characteristics, use the >ED command.
9.

To find a string, use the >FI command. To move or copy lines use the >MO or >CO commands. To copy from input tape to output tape until a specific file, use the >DU command.

The CRT input buffer is 80 characters in length. There are 10 tab points preset at 8 character intervals. Thus, the first tab point is at the 8-th column, the second at the 16-th column, etc. Entry of control I (^I) will result in a movement to the next tab point. When inputting, the cursor may not position exactly at the tab point but will position properly when the text file is outputted via the >PR command.

Text may be entered more easily by use of the auto line numbering feature (>AU command). Any >AU x where x does not equal 0 puts the TED in the auto line number mode. To temporarily exit from this mode, type >/. To prevent auto line numbering from reoccurring every time you insert or delete, enter >AU 0.

When entering source for the assembler, one need not space over to line up the various fields. Labels are entered immediately after the line number or > when in auto line numbering. Separate each source field with one or more spaces. If the format feature is set (see >FO command), the TED will automatically line up the fields. Note: If a space is entered before the label, the TED will line up the label in the next field. This should result in an assembler error when assembled. If a control I (tab) is entered, a tab to the 8-th column is performed. These tabs are preset and can not be changed. Commands, mnemonics, and pseudo ops may be entered as upper case or lower case characters. Assembly labels may also be entered in upper or lower case but a label entered as upper case will be unique to the same label entered as lower case.

3. ASSEMBLER (ASSM) FEATURES

The ASSM scans the source program in the text file. This requires at least two passes (or scans). On the first pass, the ASSM generates a label file (or symbol table) and outputs any errors that may occur. On the second pass the ASSM creates a listing and/or object file using the label file and various other internal labels.

A third pass (via >OU) may be performed in order to generate a relocatable object file of the program in the text file. This file is recorded on tape deck 0 (record deck) and may be loaded into the memory using the relocating loader at practically any location.
A. Source Statement Syntax

Each source statement consists of 5 fields as described below:

`line number label mnemonic operand comment`

**label**

The first character of a label may be formed from the following characters:

© A thru Z [ \ ] ^ _

While the remaining characters which form the label may be constructed from the above characters and the following characters:

./ 0 thru 9 : ; < > ?

The label is entered immediately after the line number or prompter (>) if in the auto line numbering mode.

**Mnemonic or Pseudo Op**

Separated from the label by one or more spaces and consists of a standard 6502 mnemonic of table A or pseudo op of table B.

**Operand**

Separated from mnemonic or pseudo op by one or more spaces and may consist of a label expression from table C and symbols which indicate the desired addressing mode from table D.

**Comment**

Separated from operand field by one or more spaces and is free format. A comment field begins one or more spaces past the mnemonic or pseudo op if the nature of such does not require an operand field. A free format comment field may be entered if a semicolon (;) immediately follows the line number or > if in auto line numbering mode.

**Note:** It is permissible to have a line with only a label. This is commonly done to assign two or more labels to the same address.

To insert a blank line, enter control I (^I).
TABLE A - 6502 Mnemonics

(For a description of each mnemonic, consult the 6502 Software Manual)

<table>
<thead>
<tr>
<th>ADC</th>
<th>CLD</th>
<th>LDA</th>
<th>SBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>CLI</td>
<td>LDX</td>
<td>SEC</td>
</tr>
<tr>
<td>ASL</td>
<td>CMP</td>
<td>LDY</td>
<td>SED</td>
</tr>
<tr>
<td>BCC</td>
<td>CPX</td>
<td>LSR</td>
<td>SEI</td>
</tr>
<tr>
<td>BCS</td>
<td>CPY</td>
<td>CLV</td>
<td>STA</td>
</tr>
<tr>
<td>BEQ</td>
<td>DBC</td>
<td>ORA</td>
<td>STX</td>
</tr>
<tr>
<td>BTC</td>
<td>DEX</td>
<td>PHA</td>
<td>STY</td>
</tr>
<tr>
<td>BMI</td>
<td>DEY</td>
<td>PHP</td>
<td>NOP</td>
</tr>
<tr>
<td>BNE</td>
<td>EOR</td>
<td>PLA</td>
<td>TAX</td>
</tr>
<tr>
<td>BPL</td>
<td>INC</td>
<td>PLP</td>
<td>TAY</td>
</tr>
<tr>
<td>BRK</td>
<td>INX</td>
<td>ROL</td>
<td>TSX</td>
</tr>
<tr>
<td>BVC</td>
<td>INY</td>
<td>ROR</td>
<td>TXA</td>
</tr>
<tr>
<td>BVS</td>
<td>JMP</td>
<td>RTI</td>
<td>TXS</td>
</tr>
<tr>
<td>CLC</td>
<td>JSR</td>
<td>RTS</td>
<td>TYA</td>
</tr>
</tbody>
</table>

TABLE B - Pseudo Ops

.BA label expression

Begin assembly at the address calculated from the label expression. This address must be defined on the first pass or an error will result and the assembly will halt.

.CT

Indicates that the source program continues on tape.

.CE

Continue assembly if errors other than 107, 104, and 117 occur. All error messages will be printed.

.LS

Set the list option so that the assembly begins printing out the source listing after the .LS on pass 2.

.LC

Clear the list option so that the assembly terminates printing the source listing after the .LC on pass 2.

.OS

Set the object store option so that object code after the .OS is stored in memory on pass 2.
.OC
Clear the object store option so that object code after the .OC is not stored in memory. This is the default option.

.WC label expression
When storing object code, move code to the address calculated from the label expression but assemble in relation to that specified by the .DA pseudo op. An undefined address results in an immediate assembly halt.

.SE label expression
Store the address calculated from the label expression in the next two memory locations. Consider this address as being an external address. Note: If a label is assigned to the .SE, it will be considered as internal.

.RC
Provide directive to relocating loader to resolve address information in the object code per relocation requirements but store code at the pre-relocated address. This condition remains in effect until a .RS pseudo op is encountered. The purpose of the .RC op is to provide the capability to store an address at a fixed location (via .SI pseudo op) which links the relocatable object code module to a fixed module.

.EJ
Eject to top of next page if >MA SET was previously entered.

.MD
Macro definition. See part 3F.

.ME
Macro end. See part 3F.

.EC
Suppress output of macro generated object code on source listing. See part 3F. This is the default state.

.ES
Output macro generated object code on source listing. See part 3F.
.DS label exp.

Define a block of storage. For example, if label exp. equated to $4$, then ASSM will skip over $4$ bytes. Note: the initial contents of the block of storage is undefined.

.RS

Provide directive to relocating loader to resolve address information in the object code per relocation, and store the code at the proper relocated address. This is the default condition.

.BY

Store bytes of data. Each hex, decimal, or binary byte must be separated by at least one space. An ascii string may entered by beginning and ending with apostrophes ('). Example: BY 00 'ABCD' 47 69 'Z' $FC $1101

.SI label expression

Store the address calculated from the label expression in the next two memory locations. Consider this address as being an internal address.

.label .DE label exp.

Assign the address calculated from the label expression to the label. Designate as external and put in label file. An error will result if the label is omitted.

.label .DI label exp.

Assign the address calculated from the label expression to the label. Designate as internal and put in label file. An error will result if the label is omitted.

.EN

Indicates the end of the source program.

Note: Labels may be entered for any of the pseudo ops.
TABLE C - Label Expressions

A label expression must not consist of embedded spaces and is constructed from the following:

**Symbolic Labels:**

One to ten characters consisting of the ascii characters as previously defined.

**Non-Symbolic Labels:**

Decimal, hex, or binary values may be entered. If no special symbol precedes the numerals then the ASSM assumes decimal (example: 147). If $ precedes then hex is assumed (example $F3). If % precedes then binary is assumed (example %11001). Leading zeros do not have to be entered. If the string is greater than 4 digits, only the rightmost 4 are considered.

**Program Counter:**

To indicate the current location of the program counter use the symbol =.

**Arithmetic Operators:**

Used to separate the above label representations:  
+ addition - subtraction

Examples of some valid label expressions follow:

```
LDA #1101 load immediate OD  
STA *TEMP+$01 store at byte following TEMP  
LDA $471E36 load from 1E36  
JMP LOOP+G- $461  
BNE =+8 branch to current PC plus 8 bytes
```

One special label expression is A, as in ASL A. The letter A followed with a space in the operand field indicates accumulator addressing mode. Thus LDA A is an error condition since this addressing mode is not valid for the LDA mnemonic.

ASL A+$00 does not result in accumulator addressing but instead references a memory location.
### TABLE D - ADDRESSING MODE FORMATS

**Immediate**
- LDA #1101 binary OD
- LDA #F3 hex F3
- LDA #F3 load value of label F3
- LDA #A ascii A
- LDA #Hi, label expression hi part of the address of the label expression
- LDA #Li, label expression lo part of the address of the label expression

**Absolute**
- LDA label expression

**Zero Page**
- LDA *label expression the asterisk (*) indicates zero page addressing

**Absolute Indexed**
- LDA label expression,X
- LDA label expression,Y

**Zero Page Indexed**
- LDA *label expression,X
- LDA *label expression,Y

**Indexed Indirect**
- LDA (label expression,X

**Indirect Indexed**
- LDA (label expression),Y

**Indirect**
- JMP (label expression)

**Accumulator**
- ASL A letter A followed with a space indicates accumulator addressing mode.


**Implied**

TAX
CLC

**Operand field ignored**

**Relative**

BEQ label expression

**B. Label File (or symbol Table)**

A label file is constructed by the assembler and may be outputted at the end of assembly (if a .LC pseudo op was not encountered) or via the >LA command. The output consist of each label encountered in the assembly and its hex address. A label in the label file which begins with a slash (/) indicates that it was defined as an external label. All others are considered as being internal labels. When a relocatable object file is generated (via >OU command), any instruction which referenced an internal label or a label expression which consisted of at least one internal label will be tagged with special information within the relocatable object file. The relocating loader uses this information to determine if an address needs to be resolved when the program is moved to another part of memory.

Conversely, instructions which referenced an external label or a label expression consisting of all external references will not be altered by the relocating loader.

At the end of the label file the number of errors which occurred and program break in the assembly will be outputted in the following format: //xxxx,yyyy,zzzz

Where xxxx is the number of errors found in decimal representation, yyyy is last address in relation to .EA, and zzzz is last address in relation to .MC.

**C. Assembling not from tape**

With the source program in the text file area, simply type >AS x. Assembly will begin starting at line number x. If a .CT pseudo is not encountered, both passes will be accomplished automatically. If a .CT pseudo op was encountered, the >PA command would have to be executed to perform the second pass.

**D. Assembling from tape**

Source for a large program may be divided into modules, entered into the text file one at a time and recorded (>PU) on tape.
At assembly, the assembler can load and assemble each module until the entire program has been assembled. This would require two passes for a complete assembly. When assembling from tape, the file identification number assigned to the modules is ignored.

Source statements within a module and the modules themselves will be assembled in the order in which they are encountered.

The ASSM assumes that if an end of file condition is encountered before the .EN pseudo op and a .CT pseudo op had not been encountered, an error is present. (at line xxxx)

When assembling from tape, the assembler should encounter a .CT pseudo op before the end of the first module. Two ways to accomplish this are:

1. a) Load the first module via the \( GE \) command.
   b) This module should contain a .CT pseudo op

   OR

2. a) Clear the text file via the \( CL \) command
   b) Enter \( 9999 \) .CT
   9999 is entered since one may have requested any assembly beginning with a line number. This insures that the .CT gets executed.

Next ready the play deck and type \( AS X \). Either way the ASSM will start and stop tape deck 1 in the assembly process until the .EN pseudo op is encountered. At that point tape deck 1 is turned off, and the message READY FOR PASS 2 is outputted.

One is now in the TED mode. Rewind the tape deck (\( ON 1 \) and \( OF 1 \) or Ti accordingly). Perform 1 or 2 as described above and type PASS to perform the second pass. Again tape deck 1 will be turned on and off accordingly under control of the ASSM software.

E. Creating a relocatable object file (\( O/U \))

In order to create a relocatable object file, the programmer should define those labels whose address should not be altered by the relocating loader. This is done via the .DE pseudo op. Non-symbolic labels (example: \( \#0169 \)) are also considered as being external. All other labels (including those defined via the .DI pseudo op) are considered as internal. Addresses associated with internal labels are altered by an offset when the program is loaded via the relocating loader.
Also, the .SE stores a two byte external address and the .SI stores a two byte internal address. Similarly the relocating loader will alter the internal address and not the external address.

An example of an external address would be the calls to your ROM monitor or any location whose address remains the same no matter where the program is located. Locations in zero page are usually defined as external addresses. Expressions consisting of internal and external labels will be combined and considered an internal address. A label expression consisting entirely of external labels will be combined and considered as external.

To record a relocatable object file, insert a blank tape in tape deck 0 and ready. If the entire source program is in memory, simply type \texttt{OU}.

If the source program is on tape, ready as described in 1 and 2 in part 3D and then type \texttt{OU}. The ASSM will turn both tape decks on and off until the end of assembly. The relocatable object file will be recorded on the tape in deck 0.

After the relocatable object file has been recorded, record an end of file mark via the \texttt{FU X} command.

F. Macros

ASSM/TED provides a macro capability. A macro is essentially a facility in which one line of source code can represent a function consisting of many instruction sequences. For example, the 6502 instruction set does not have an instruction to increment a double byte memory location. A macro could be written to perform this operation and represented as \texttt{INCD (VALUE,1)}. \textbf{This macro} would appear in your assembly language listing in the mnemonic field similar to the following:

```
BNE SKIP
NOP

\{  

INCD (VALUE,1); INCREMENT DOUBLE
LDA TEMP

\}
```

Before a macro can be used, it must be defined in order for ASSM to process it. A macro is defined via the \texttt{.MD} (macro definition) pseudo op. Its form is:

```
!!label .MD (L1 L2 ... Ln)
```

Where label is the name of the macro (!! must precede the label), and L1, L2, ..., Ln are dummy variables used for replacement with the expansion variables. These variables should be separated using spaces, do not use commas.
To terminate the definition of a macro, use the .ME (macro end pseudo op).

For example, the definition of the INCD (increment double byte) macro could be as follows:

```assembly
!INCD .MD (LOC) ;INCREMENT DOUBLE
INC LOC
BNE SKIP
INC LOC+1
.SKIP
.ME
```

This is a possible definition for INCD. The assembler will not produce object code until there is a call for expansion. Note: A call for expansion occurs when you enter the macro name along with its parameters in the mnemonic field as INCD (TEMP) or INCD (COUNT) or INCD (COUNT+2) or any other labels or expressions you may choose.

Note: In the expansion of INCD, code is not being generated which increments the variable LOC but instead code for the associated variable in the call for expansion.

If you tried to expand INCD as described above more than once, you will get a !06 error message. This is a duplicate label error and it would result because of the label SKIP occurring in the first expansion and again in the second expansion.

There is a way to get around this and it has to do with making the label SKIP appear unique with each expansion. This is accomplished by rewriting the INCD macro as follows:

```assembly
!INCD .MD (LOC) ;INCREMENT DOUBLE
INC LOC
BNE ...SKIP
INC LOC+1
...SKIP .ME
```

The only difference is ...SKIP is substituted for .SKIP. What the ASSM does is to assign each macro expansion a unique macro sequence number (2**16 maximum macros in each file). If the label begins with ... the ASSM will assign the macro sequence number to the label. Thus, since each expansion of this macro gets a unique sequence number, the labels will be unique and the !06 error will not occur.

If the label ...SKIP also occurred in another macro definition, no !06 error will occur in its expansion if they are not nested. If you nest macros (i.e. one macro expands another), you may get a !06 error if each definition uses the ...SKIP label.
The reason this may occur is that as one macro expands another in a nest, they each get sequentially assigned macro sequence numbers. As the macros work out of the nest, the macro sequence numbers are decremented until the top of the nest. Then as further macros are expanded, the sequence numbers are again incremented. The end result is that it is possible for a nested macro to have the same sequence number as one not nested or one at a different level in another nest. Therefore if you nest macros, it is suggested that you use different labels in each macro definition.

Some further notes on macros are:

1) The macro definition must occur before the expansion.

2) The macro definition must occur in each file that references it. Each file is assigned a unique file sequence number (2**16 maximum files in each assembly) which is assigned to each macro name. Thus the same macro can appear in more than one file without causing a !06 error. If a macro with the same name is defined twice in the same file, then the !06 error will occur.

3) Macros may be nested up to 32 levels. This is a limitation because there is only so much memory left for use in the stack.

4) If a macro has more than one parameter, the parameters should be separated using spaces - do not use commas.

5) The number of dummy parameters in the macro definition must match exactly the number of parameters in the call for expansion.

6) The dummy parameters in the macro definition must be symbolic labels. The parameters in the expansion may be symbolic or nonsymbolic label expressions.

7) If the .ES pseudo op is entered, object code generated by the macro expansion will be output in the source listing. Also, comment lines within the macro definition will be output as blank lines during expansion. Conversely, if .EC was entered, only the line which contained the macro call will be output in the source listing.

8) A macro name may not be the same as a 6502 mnemonic, pseudo op, or conditional assembly operator.
G. Conditional Assembly

ASSM also provides a conditional assembly facility to conditionally direct the assembler to assemble certain portions of your program and not other portions. For example, assume you have written a CRT controller program which can provide either 40,64 or 80 characters per line. Instead of having to keep 3 different copies of the program you could use the ASSM conditional assembly feature to assemble code concerned with one of the character densities.

Before we continue with this example, let's describe the Conditional assembly operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>***</td>
<td>Three asterisks in the mnemonic field indicates the end of the control block.</td>
</tr>
<tr>
<td>IFE label exp.</td>
<td>If the label expression equates to a zero quantity, then assemble to end of control block.</td>
</tr>
<tr>
<td>IFN label exp.</td>
<td>If the label expression equates to quantity not equal to zero, then assemble to end of control block.</td>
</tr>
<tr>
<td>IFF label exp.</td>
<td>If the label expression equates to a positive quantity (or 0000), then assemble to end of control block.</td>
</tr>
<tr>
<td>IFM label exp.</td>
<td>If the label expression equates to a negative (minus) quantity, then assemble to end of control block.</td>
</tr>
</tbody>
</table>

Note: All label expressions are equated using 16-bit precision arithmetic.
Going back to the CRT controller software example, a possible arrangement of the program is as follows:

```
CHAR.LINE .DE 40
{
    IFE CHAR.LINE-40
    ;CODE FOR 40 CHAR./LINE
    {
        ***
        IFE CHAR.LINE-64
        ;CODE FOR 64 CHAR./LINE
        {
            ***
            IFE CHAR.LINE-80
            ;CODE FOR 80 CHAR./LINE
            {
                ***
            }
        }
    }
    ;COMMON CODE
    {
    }
}
```

Shown is the arrangement which would assemble code associated with 40 characters per line since CHAR.LINE is defined as equal 40. If you wanted to assemble for 80 characters, simply define CHAR.LINE as equal 80.

Conditional assembly can also be incorporated within macro definitions. A very powerful use is with a macro you don't want completely expanded each time it is referenced. For example, assume you wrote a macro to do a sort on some data. It could be defined as follows:

```
EXPAND .DE 0
|||SORT .MD
    IFN EXPAND
        JSR SORT.CALL ;CALL SORT
    ***
    IFE EXPAND
        JSR SORT.CALL
        JMP ...ABC
;SORT CODE FOLLOWS
SORT.CALL
{
     RTS
...
...ABC SET EXPAND=1

```
In this example, EXPAND is initially set to 0. When the macro is expanded for the first time, EXPAND equals 0 and the code at SORT:CALL will be assembled along with a JSR to and a JMP around the sort subroutine. Also the first expansion sets EXPAND to 1. On each succeeding expansion, only a JSR instruction will be assembled since EXPAND equals 1. Using conditional assembly in this example resulted in more efficient memory utilization over an equivalent macro expansion without conditional assembly.

H. Default Parameters on entry to ASSM

. Assumes not assembling from tape (otherwise use .CT)
. Does not store object code in memory (otherwise use .OS)
. Begins assembly at $0200 (otherwise use .BA)
. Output listing set (otherwise use .LC)
. Stops assembly on errors (otherwise use .OE)
. Stores object code beginning at $0200 unless a .BA or .MC is encountered and if .OS is present.
. Object code generated by macros does not appear on the assembly listing (i.e. default is .EC).
4. EXAMPLES

A. Listing illustrating text entry

An example of the printout which occurs when inputting text in the text file follows:

```
>FORMAT SET
>AUTO 10
>100;THIS PROGRAM ADDS 06 TO REGISTER X
0110>START TXA
0120> CLC
0130> ADC #6
0150>END RTS
0160> .EN
0170> //
0174> TAX
0151> //
```

Note the use of // to terminate the auto line numbering. Auto line numbering can be restarted by simply entering the line number where insertion is to begin. To prevent auto line numbering, simply type >AU or >AU 0.

B. Output listing from ASSM

Listing 1 is a source listing output of a program which provides formatted hex dump of a block of memory. It is presently configured for TIM based systems but can be easily modified for other systems.

5. USING THE RELOCATING LOADER

A source listing of the relocating loader (listing 2) is provided. The relocating loader is not part of the ASSM/TED program body, and the user will have to enter it via the listing.

If you prefer to have the loader to reside in some other part of memory, you should enter the source into the text file, assemble, and then create a relocatable object file on tape.

To record a program in relocatable format, first assemble (without a .OS pseudo op) the program at location 0000 (.BA $0). Next create a relocatable object file via the >OU command. Terminate the relocatable object file with an end of file mark via the >PU X command. To reload a program in relocatable format, first enter the address where you want the program to reside in memory locations O0E0 (l0) and O0E1 (h1), the modules file number in O110, and then execute.
When executing the relocating loader, if an error or an end of file mark is detected, a break (BRK) instruction will be executed so as to return to your monitor. The contents of register A indicates the following:

OO good load
EE error in loading

All programs to be created in relocatable format should be assembled at $0000. This is because the offset put in 00EO and 00EI before execution is added to each internal address by the loader in order to resolve addresses while relocating the program. If the program was originated at say 1000, then one would have to enter F200 as the offset in order to relocate to 0200 (i.e. F200+1000 = 0200). This is somewhat more confusing than an assembly beginning 0000.

In addition to the program memory space, the relocating loader uses the following memory locations.

0008-00C9, 00DC-00E1
0110, 011E-0121, 017A-0184

Plus other stack area for subroutine control.

6. CONFIGURE ASSM/TED FOR DISC OPERATION

ASSM/TED provides the user with four 2-byte address vectors for linkage to your disc operating system (DOS). They are:

<table>
<thead>
<tr>
<th>DISC1</th>
<th>$F0, $F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address vector to your DOS (or patch to DOS) which accepts the output data filename beginning at $0135, Y. The user provided patch should accept filename characters by incrementing R(Y) until a space is encountered. If R(Y)=50 hex then your DOS should instead treat this as a CLOSE output file operation.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISC2</th>
<th>$F2, $F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address vector to your DOS (or patch to DOS) which accepts the input data file name beginning at $0135, Y. The user provided patch should accept filename characters by incrementing R(Y) until a space is encountered. If R(Y)=50 hex then your DOS should instead treat this as a CLOSE input file operation.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISC1.VEC</th>
<th>$F6, $F7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector to your DOS (or patch) indicating that data is to be conditionally loaded into memory defined as follows:</td>
<td></td>
</tr>
</tbody>
</table>
LOAD/WO - if=1 then enter in memory.
($123$) if=0 then get from disc but don't move to memory.
This is required to skip over files not selected.

START.ADD - start address of memory.
($124$-$125$)

END.ADD - end address of memory.
($126$-$127$)

DISCO.VEC $\#4$, $\#5$
Vector to your DOS (or patch) indicating that data in memory range
START.ADD thru END.ADD is to be stored on disc. LOAD/WO should
be ignored.

7. USING ASSM/TED WITH DISC

Before operating with the disc, the user should set up the
address vectors as described in part 6. This could be done by
executing user provided code using the >RUN command, or simply
manually entering address vectors using your system monitor.

There are two commands which determine if data is to input or
output from tape or disc. They are:

>ENTER

Enter in disc directory. A vector thru DISC1 is performed. If
entered with a filename then an open of the output file is
performed. At this point all output normally going to tape will
go through vector DISCO.VEC. If no parameters are entered, when
your DOS should assume a close operation. At this point any
output will be to tape.

>LOOKUP

Lookup in disc directory. A vector thru DISC2 is performed. If
entered with a filename then an open of the input file is
performed. At this point all input normally read from tape will
go through vector DISC1.VEC. If no parameters are entered, then
your DOS should assume a close operation. At this point any input
will be from tape.
8. ERROR CODES

An error message of the form \( \text{xx AT LINE yyy/zz} \) where xx is the error code, yyy is the line number, and zz is the file number will be outputted if an error occurs. Sometimes an error message will output an invalid line number. This occurs when the error is on a non-existent line such as an illegal command input.

The following is a list of error codes not specifically related to macros:

**ERROR CODE**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Checksum error on tape load.</td>
</tr>
<tr>
<td>16</td>
<td>Illegal tape deck number.</td>
</tr>
<tr>
<td>15</td>
<td>Syntax error in ( \text{'&gt;ED} ) command.</td>
</tr>
<tr>
<td>12</td>
<td>Command syntax error or out of range error.</td>
</tr>
<tr>
<td>11</td>
<td><strong>Missing parameter in ( \text{&gt;'+NU} ) command.</strong></td>
</tr>
<tr>
<td>10</td>
<td>Overflow in line # renumbering. CAUTION--You should properly renumber the text file for proper command operations.</td>
</tr>
<tr>
<td>09</td>
<td>Overflow in text file - line not inserted.</td>
</tr>
<tr>
<td>08</td>
<td>Overflow in label file - label not inserted.</td>
</tr>
<tr>
<td>07</td>
<td>Expected hex characters, found none.</td>
</tr>
<tr>
<td>06</td>
<td>Illegal character in label.</td>
</tr>
<tr>
<td>05</td>
<td>Unimplemented addressing mode.</td>
</tr>
<tr>
<td>04</td>
<td>Error in or no operand.</td>
</tr>
<tr>
<td>03</td>
<td>Found illegal character in decimal string.</td>
</tr>
<tr>
<td>02</td>
<td>Underfined label (may be illegal label).</td>
</tr>
<tr>
<td>01</td>
<td>EN pseudo op missing.</td>
</tr>
<tr>
<td>00</td>
<td>Duplicate label.</td>
</tr>
<tr>
<td>00</td>
<td>Label missing in ( \text{'.DE} ) or ( \text{'.DI} ) pseudo op.</td>
</tr>
<tr>
<td>00</td>
<td>.BA or ( \text{'.MO} ) Operand Undefined.</td>
</tr>
<tr>
<td>00</td>
<td>Illegal pseudo op.</td>
</tr>
<tr>
<td>00</td>
<td>Illegal mnemonic.</td>
</tr>
<tr>
<td>00</td>
<td>Branch out of range.</td>
</tr>
<tr>
<td>00</td>
<td>Not a zero page address.</td>
</tr>
<tr>
<td>00</td>
<td>Error in command input.</td>
</tr>
</tbody>
</table>


The following is a list of error codes that are specifically related to macros and condition assembly:

**ERROR CODE**

2F  Overflow in file sequence count (2**16 max.)
2E  Overflow in number of macros (2**16 max.)
2B  .ME without associated .MD
2A  Non symbolic label in SET
29  Illegal nested definition
27  Macro definition overlaps file boundary
26  Duplicate macro definition
25  Quantity parms mismatch or illegal characters
24  Too many nested macros (32 max.)
23  Macro definition not complete at .EN
22  Conditional suppress set at .EN
21  Macro in expand state at .EN
20  Attempt expansion before definition

9. FILE NUMBERS

Information to be recorded on tape via the \$PU and \$OU commands may be assigned a file identification number to distinguish between other files of information. A file number is a decimal number between 0 and 99. To enter a file number as a parameter in the \$PU, \$OU, or \$GE commands, begin with the letter 'F' followed by the file number. Examples are F0, F17, F6, etc. If no file number is entered with the \$PU and \$OU commands, file number 0 will be assigned by default.

When loading, all files encountered will result in the outputting of their associated file numbers and file length in bytes. The loaded file has, in addition, the memory range of the location of the loaded data. Example: \$GET F17
\$P00 01A3
F67 0847
F17 0F93 0200-1193

An end of file mark may be recorded via the \$PU X command to indicate the end of a group of files. If an end of file mark is encountered when loading, FEE will be outputted and a return to the command mode will be performed.
10. STRING SEARCH AND REPLACE COMMANDS

A. Edit command

A powerful string search and replace, and line edit capability is provided via the \texttt{EDIT} command to easily make changes in the text file. Use form 1 to string search and replace, and form 2 to edit a particular line.

Form 1

\texttt{\textbackslash EDIT tSlsS2t \%d \* x y}

Where: \( t \) is a non-numeric, non-space terminator

\( S1 \) is string to search for.

\( S2 \) is string to replace \( S1 \).

\( d \) is don't care character. Proceed with \% character to change the don't care, else don't care character will be \% by default.

\* indicates to interact with user via subcommands before replacing \( S1 \).

\# indicates to alter but provide no printout.

\( \Delta \) (space) indicates to alter and provide printout.

\( x \) line number start in text file.

\( y \) line number end in text file.

Asterisk (*) prompted subcommands:

- \( A \): alter field accordingly.
- \( D \): delete entire line.
- \( M \): move to next field - don't alter.
- \( S \): skip line - don't alter.
- \( X \): exit \texttt{EDIT} command
- \( \texttt{\%F} \): (control F) - enter form 2

Defaults:

- \( d = \% \)
- \( x = 0 \)
- \( y = 9999 \)
- \( \Delta = \text{space} \)

Print all lines altered

For example, to replace all occurrences of the label LOOP with the label START between lines 100 and 600, enter:

\texttt{\textbackslash EDIT .LOOP.START 100 600}

To simple delete all occurrences of LOOP, enter:

\texttt{\textbackslash EDIT .LOOP.. 100 600}

Use the *,\#, and \( \Delta \) as described.

The period was used in the above examples as the terminator but any non-numeric character may be used.

At the end of the \texttt{EDIT} operation, the number of occurrences of the string will be output as \texttt{/xxyy} where \texttt{xxyy} is a decimal quantity.
Form 2

>EDIT n

Where: n is line number (0-9999) of line to be edited.

subcommands: ^P (control P) - Find user specified character.
              ^C (carriage return) - retain any remaining part of a line.
              ^D (control D) - delete any remaining part of line.
              ^H delete a character.

For example, to change LDA to LDY in the following line

LOOP1  LDA #L,CRTBUFFER LOAD FROM BUFFER

Type ^P followed with A, then ^H, then Y, and then terminate line with a carriage return.

The corrected line will then be outputted and entered in the text file.

B. Find Command

If you want to just find certain occurrences of a particular string, use the >FIND command. Its form is:

    >FIND tSlit # x y

Where: t, Sl, #, x, y are as defined in part 10.A.

For example, >FIND /LDA/ will output all occurrences of the string LDA in the text file.

AT the end of the >FIND operation, the number of occurrences of the string will be output as //xxxx where xxxx is a decimal quantity.

A unique use of this command is to count the number of characters in the text file (excluding line numbers). The form for this is:

    >FIND //#

11. CONTROL CODES

Ascii characters whose hex value is between hex 00 and 20 are normally non-printing characters. With a few exceptions, these characters will be output in the following manner: ^c where c is the associated printable character if hex 40 was added to its value. For example, ascii 03 will be output as ^C, 18 as ^X, etc.

In addition, some of these control codes have special functions in ASSM/TED.
Control codes which have special functions are:

^@          * null (hex00)
^B          * go to Basic
^C          * go to Monitor (executes BRK instruction)
^D          * delete - used by >EDIT
^F          * find - used by >EDIT
^G          * bell
^H          * backspace (delete previous character)
^I          * horizontal tab
^J          * linefeed
^M          * carriage return
^O          * continue processing but suppress output to CRT
^Q          * continue after break operation
^T          * (as ^Tn) toggle Motor Control on deck n
^X          * delete entire line entered
^Y          * jump to location $0000. Return via warm start
^Z          * terminate processing and go to ">" mode.
^[          * escape character

*= Non-printing control character

12. SPECIAL NOTES

. In addition to the program memory space the ASSM/TED uses
  the following memory locations

  0100 - up depending on type of function
  00B9 - OOP8

  plus other stack area for subroutine control. The CRT
  buffer is in locations  0135 - 0185

. Keep the cover closed on the tape deck as this keeps
  the cassette cartridge stable.

. When entering source modules (without .EN) you can perform
a short text on the module by assembling the module while
in the text file and looking for the IO7 error. If other
error messages occur, you have errors in the module. This
short text is not a complete test but does check to make
sure you have lined up the fields properly, not entered
duplicate labels within the module, or entered illegal
mnemonics or addressing modes.
A 64 character/line (or greater) output device should be used with this program when printing an assembly listing in order to provide a neat printout without foldover to next line.

Any keyboard input greater than 80 characters in length will be automatically inserted in the text file without the user having to enter a carriage return.

Locations $00D5 (lo) and $00D6 (hi) contain the address of the present end of the label file. This address +2 should contain a zero (a forward pointer).

Locations $00D3 (lo) and $00D4 (hi) contain the address of the present end of the text file. This address +2 should contain a zero (a forward pointer).

The ASSM/TED and the Relocating Loader were designed so that they will execute in RAM or ROM.

To find the address of an entry in the text file, output the line via the PR command, issue the BR command, and then get the contents of memory location OODD, OODE. This is an address which points to the end of the outputted line.

**LISTINGS**

1. Hex dump program
2. Source listing of relocating loader

**TABLES**

A) 6502 Mnemonics
B) Pseudo ops
C) Label expression
D) Addressing Modes
LISTING 1

0100 THIS PROGRAM IS PROVIDED AS AN EXAMPLE OF A PROGRAM
0200 WHICH USES VARIOUS FEATURES DESCRIBED IN THIS MANUAL.
0300 THIS PROGRAM OUTPUTS A HEX LISTING

0400 1
0500 DA $0
0600 DC
0700 CRFLF .DE $728A
0800 TBYT .DE $7281
0900 SPACE .DE $7377
1000 SPACE2 .DE $7374
1100 COUNT .DI END+DF+PGM
1200 ADDRS .DE $0
1300 END .DE $010A
1400 
1500 IAT START, SET PRINTER TO BEGIN PRINTING ON 3-TH LINE
1600 IO 3-TH LINE
1700 ISTART ADDRESS IN ADDRS
1800 END ADDRESS IN END
1900 1
2000 1
2100 MACRO DEFINITION -- INCREMENT DOUBLE BYTE
2200 LE
2300 
2400 !!!!INCD .MD (X)
2500 INC *X
2600 BNE ...SKIP
2700 INC *X+1
2800 ...SKIP .ME
2900 
3000 1

0000- A9 00 3100 BEGIN LDR #$00
0002- 8A 00 3200 TAC
0003- 8D 5B 00 3300 STR COUNT
0006- 20 8A 72 3400 NEXTLN LSR CRFLF
0009- AD 5B 00 3500 LDR COUNT
000C- C9 3C 3600 IDEC. 60 LINES PER PAGE
3700 CMP #$3C 1DECIMAL 60
000E- 90 0D 3800 BCC SKIP
0010- A9 00 3900 LDR #$00
0012- 8D 5B 00 4000 ISSUE 6 CRFLF'S AT END OF 60-TH LINE TO GO
4100 TO NEXT PAGE
4200 1TO NEXT PAGE
0015- A0 06 4300 LDY #$06
0017- 20 8A 72 4400 LOOP3 LSR CRFLF
001A- 88 4500 BNE
001B- 00 FA 4600 BNE LOOP3
001D- A0 10 4700 SKIP LDY #$10
001F- A5 01 4800 LDR +ADRS+1
0021- 20 B1 72 4900 JSR TBYT
0024- A5 00 5000 LDR +ADRS+0
0026- 20 B1 72 5100 JSR TBYT
0029- 20 74 73 5200 JSR SPACE2
5300 INDX ADDRESS IS OUTPUTED
002C- A1 00 5400 LOOP2 LDR (ADRS+X)
002E- 20 B1 72 5500 JSR TBYT
0031- AS 01  5600     LDA $ADDRES+$01
0033- CD 0B 01  5700   CMP END+$1
0036- 90 11  5800   BCC NOT+END
0038- F0 08  5900   BEO CKLO
003A- 20 8A 72  6000  END+PGM  JSR CRLF
003D- 00   6100   BRK
003E- EA 6200   NOP
003F- 4C 00 00  6300   JMP BEGIN
0042- A5 00  6400 CKLO  LDA $ADDRES+$0
0044- CD 0A 01  6500   CMP END+$0
0047- B0 F1  6600   BCS END+PGM
0049- 6700 NOT+END  INCD (ADDRES) ; INCREM. ADDRS
004E- E6 00  
004F- D0 02  
0050- E6 01  
0052- 20 77 73  6800  JSR SPACE
0055- 88 6900   DEY IR(Y)=BYTE COUNTER
0058- EE D7  7000   BNE LOOP2
005B- EE 5B 00  7100   INC COUNT
005E- 4C 06 00  7200   JMP NEXT+LN
0061- 7300 END+DF+PGM .EN

LABEL FILE: [ / = EXTERNAL ]

/CRLF=728A /TBYT=72B1 /SPACE=7374
/SPACE=7374 /COUNT=005B /ADDRS=0000
/END=010A /BEGIN=0000 NEXT+LN=0006
/LOOP=0017 /SKIP=0010 LOOP2=002C
END+PGM=003A /CKLO=0042 NOT+END=0049
X=0000  END+DF+PGM=005B
/0000,005B,005B
PAGE 01

0060  .EJ
0070  **********************************************
0080  * COPYRIGHT 1980 BY U.D.SCHRODER *
0090  *
0100  *
0110  **********************************************
0120  *
0130  *PURPOSE OF THIS PART:*
0140  11. ASSIST YOU TO IMPLEMENT THE ASSM/TED ON YOUR SYSTEM
0150  12. WARN YOU ON SOME DIFFICULTIES THAT MIGHT OCCUR
0160  1 USING THE ASSM/TED
0170  *
0180  1
0190  1

3C27- 4C E8 3F
0200  JMP CRT/OUTPUT
0210  1
0220  1
0230  .BA $3FD6
0240  .CRT/OUTPUT
0250  1PURPOSE: OUTPUT A CHARAKTER TO THE OUTPUTDEVICE
0260  1THE ASCII-CHARAKTER IS IN A.
0270  1AT RETURN THE CONTENT OF ALL REGISTERS
0280  1MAY BE CHANGED.
0290  1

3F6E- A8 1F
0300  1CRT/CRCRND LDA #1F 1DELAYTIME AFTER CR
0310  JSR DELAY
3FDB- A9 0A 3F
0320  LDA #1OA 1OUTPUT LNFD AFTER CR
0330  JSR CRT/OUTPUT
3FDD- A9 EB 3F
0340  LDA #1EB 1DELAYTIME AFTER LNFD
3FE2- A9 07 3F
0350  JSR DELAY
3FE5- EA
0360  NOP
3FE6- EA
0370  NOP
3FE7- EA
0380  RTS
3FE8- A0 30 3F
0390  CRT/OUTPUT PHA
3FE9- EA
0400  NOP 1INSERT LDY #01F
3FEA- EA
0410  NOP 1 BNE CRT/OUT2
3FEB- EA
0420  NOP 1IF YOU USE ONLY ONE
3FEC- EA
0430  NOP 1HARDCOPY DEVICE FOR INPUT/OUTPUT
3FED- EA
0440  NOP
3FEE- A0 A0 3E
0450  JSR KIM/OUTCH
3FF1- EA
0460  CRT/OCT2 PLA
3FF2- EA
0470  CMP #0D 1CARRIAGE RETURN NEEDS
3FF4- EA
0480  BEQ CRT/CRCRND
3FF5- EA
0490  RTS
0500  1

0510  1
0520  1KEYBOARD INPUT
0530  1THE KIMVERSION ASSUMES THAT INPUT WILL
0540  1BE ECHOED BY HARDWARE. IF YOU CAN PREVENT
0550  1THE ECHO. PLEASE INSERT JSR $3BF1 AT 3CF
0560  .BA $3C69
0570  JSR KIM/GETCH 1RETURN CHAR IN (A)
0580  1
0590  .BA $3C6A
0600  1

3CB9- 20 5A 1E
0610  JSR KIM/GETCH 1RETURN CHAR IN (A)
0620  1
0630  1

3C9F- EA
0640  NOP 1CHANGE THIS INTO JSR $3BF1
3C90- EA
0650  NOP 1IF IT IS POSSIBLE FOR YOU
3CA1- EA
0660  NOP 1TO PREVENT ECHOING THE INPUTCHAR.
THE BREAK-TEST ROUTINE IS ENTERED ONLY IMMEDIATELY AFTER PRINTING A CARRIAGE RETURN.

IF YOU DON'T WAIT ON THE PROMPTING CHARACTER AND ENTER ANY KEY WHILE THE CARRIAGE RETURN IS PRINTED, THAT KEY WILL BE INTERPRETED AS A BREAK!!! EVERY KEY OR COMMAND THEREAFTER WILL BE IGNORED - EXCEPT SOME CONTROL KEYS!!! TO RECOVER PRESS CONTROL-O OR CONTROL-Z UNTIL YOU GET THE PROMPTING CHARACTER AGAIN.

..................

.BA $3BD2 1ONLY ONE CALL!!!

.JSR BREAKTEST 1CARRY=1=BREAK

.BA $3F77

.BREAKTEST

PURPOSE: DETECT IF ANY KEY IS DOWN WHILE PRINTING CARRIAGE-RETURN

EXPLANATION: INPUT ON KIM IS ON THE MOST SIGNIFICANT BIT OF $1740.

BREAKTEST BIT $1740

CLC 1CARRY=0=NO BREAK

BMI NO/BREAK 1MSB=1=NO INPUT

BREAK/WAIT BIT $1740 1WAIT UNTIL END OF BREAK

BPL BREAK/WAIT

JSR KIM/GETCH 1IGNORE KEY AFTER BREAK

SEC 1CARRY=1=BREAK

NO/BREAK RTS

..................

DELAY TIME ACCORDING TO CONTENT OF A. DURATION IS (A)*(A)*(A), WITH 2 (A <$FF)

USED ONLY BY CRT/CR,FND

DELAY PHA

DELAY,2 PHA

SBC #$1

BNE DELAY,4

PLA

SBC #$1

BNE DELAY,2

PLA

SBC #$1

BNE DELAY

RTS

KIM/GETCH .DE $1E5A 1USED TWICE

KIM/OUTCH .DE $1EA0 1USED ONE TIME
1210      .EJ
1220      'THE BREAK IS USED ONCE AT 20A9 TO EXIT TED/ASSM
1230      'IF YOU DON'T LIKE THIS WAY OF EXIT
1240      'CHANGE THIS BREAK INTO A JUMP TO ???
1250      ;
1270      'ADDRESS WHERE A SUBROUTINE IS CALLED
1280      'TO INIT THE BREAK-VEKTOR ON THE KIM
1290      .BA #202B
202B- 20 A6 3F
1300      JSR BRK.VCT.IN IT
1310      ;
1320      .BA #3FA6
3FA6- A9 00 1330      BRK.VCT.IN LDA #1000 1#1000 IS BREAK-ENTRY ON KIM
3FA6- 8D FE 17 1340      STA #17FE 1KIM DOES JMP (#17FE) AT BREAK
3FAD- A9 1C 1350      LDA #11C
3FAD- 8D FF 17 1360      STA #17FF
1370 ;
1380      'INIT OF I/O PORT FOR CASSETTE
1390 ;
3FB0- AD 03 17 1400      LDA #1703
3FB3- 09 0B 1410      ORA #00001011
3FB5- AD 03 17 1420      STA #1703
3FB6- E0 1430      RTS
1440 ;
1450 ;
1460 ;
1470 ;
1480 ;TABLE OF MEMORY-USE AT 362C
1490 ;
1500      .BA #362C
1510 ;
1520 ;.SI TEXTBUF/START
1530 ;.SI TEXTBUF/END 1ASSM/TED USES 3 BYTES MORE!!
1540 ;
1550 ;.SI SYMTAB/START
1560 ;.SI SYMTAB/END 1ASSM/TED USES 3 BYTES MORE
1570 ;
1580 ;.SI RELOC/BUFF 1256 BYTE BUFFER
1590   \texttt{.EJ}
1600   \texttt{\texttt{1EXA}MPLES OF HARDCOPY DRIVER FOR PRINTER.}
1610   \texttt{\texttt{1NOTE: FOR EACH NEW LINE THE SUBROUTINE HAS TO RETURN}}
1620   \texttt{\texttt{1ONE BYTE \#0A FOR THE LINUECOUNTER ROUTINE.}}
1630   \texttt{\texttt{1REGISTERS X, Y MAY BE CHANGED.}}
1640   \texttt{\texttt{1}}
1650   \texttt{\texttt{1HARDCOPY .DE \#0000 1FOR REFERENCE ONLY}}
1660   \texttt{\texttt{1}}
1670   \texttt{\texttt{1}}
1680   \texttt{\texttt{1}}
1690   \texttt{\texttt{1}}
1700   \texttt{\texttt{1PRINTER WITH AUTO-LINEFEED}}
1710   \texttt{\texttt{1}}
1720   \texttt{\texttt{38D4\texttt{- 20 68 41 1730 1JSR HARDCOPY/A}}}
1740   \texttt{\texttt{1750 1\texttt{BA \#4160 1OR ELSEWHERE}}}
1760   \texttt{\texttt{4168\texttt{- 48 1770 1JSR HARDCOPY}}}
1780   \texttt{\texttt{4169\texttt{- 20 00 00}}}
1790   \texttt{\texttt{416C\texttt{- 68 1780 1PLA}}}
1800   \texttt{\texttt{416F\texttt{- C9 0D 1800 1BNE HC/RTS 1DN CARRIAGE RETURN}}}
1810   \texttt{\texttt{4171\texttt{- A9 0A 1810 1LDA \#00A 1CHANGE \#0D INTO \#0A}}}
1820   \texttt{\texttt{4173\texttt{- 60 1820 1HC/RTS RTS}}}
1830   \texttt{\texttt{END/A}}
1840   \texttt{\texttt{1}}
1850   \texttt{\texttt{1CHANGE START OF TEXTBUFFER}}
1860   \texttt{\texttt{1870 1\texttt{BA \#362C}}}
1880   \texttt{\texttt{1890 1}}
1900   \texttt{\texttt{1910 1\texttt{PRINTRODUCTION WITH AUTO-LINEFEED}}}
1920   \texttt{\texttt{1930 1PRINTER WITHOUT AUTO-LINEFEED}}
1940   \texttt{\texttt{1950 1\texttt{BA \#38D4}}}
1960   \texttt{\texttt{1970 1JSR HARDCOPY/B}}
1980   \texttt{\texttt{1990 1\texttt{BA \#4160 1OR ELSEWHERE}}}
2000   \texttt{\texttt{4168\texttt{- A9 0A 1930 1D0/LINEFEED LDA \#00A}}}
2010   \texttt{\texttt{416A\texttt{- 48 2000 1HARDCOPY/B PHA}}}
2020   \texttt{\texttt{416B\texttt{- 20 00 00 2010 1JSR HARDCOPY}}}
2030   \texttt{\texttt{416E\texttt{- 68 2020 1PLA}}}
2040   \texttt{\texttt{416F\texttt{- C9 0D 2030 1CMP \#0D 1DN CARR.RET.: ADD LNFD}}}
2040   \texttt{\texttt{4171\texttt{- F0 F5 2040 1BNE DO/LINEFEED}}}
2050   \texttt{\texttt{4173\texttt{- 60 2050 1RTS}}}
2060   \texttt{\texttt{END/B}}
2070   \texttt{\texttt{2080 1CHANGE START OF TEXTBUFFER}}
2090   \texttt{\texttt{2100 1\texttt{BA \#362C}}}
2100   \texttt{\texttt{362C\texttt{- 74 41}}}
2120   \texttt{\texttt{2130 1}}
MODIFICATIONS IN THE CASSETTE ROUTINES SUPPLIED

CHANGED IN/PORT ROUTINE OF CASSETTE PROGRAM

IN/PORT
        LDA C/PORT
        AND #FF
        AND #%00000100
        RTS

SOFTWARE CAN INVERT TO!!

1. DO NOT CONNECT THE REMOTE-CONTROL OR EARPHONE
2. IT MIGHT DESTROY THE PIA!!!
3. USING BITS OF THE B-PORT AS OUTPUT - NOTE THIS!
4. READING A OUTPUTBIT WILL NOT ALWAYS REFLECT THE
5. ACTUAL VALUE THAT IS OUTPUTTED. READING THE
6. B-PORT RETURNS THE VALUE THAT IS READ ON
7. THE OUTPUT PINS - SO EXCESSIVE LOADING A PORT
8. TO GROUND WILL ALWAYS RETURN ZERO.
9. CONCLUSION: DO NOT EXCESSIVE LOAD ANY BIT.

LABEL FILE: + / = EXTERNAL +

CRT/CR,FND=3F86 CRT/OUTPUT=3FEB CRT/OUT2=3FF1
BREAK=3F77 BREAK/1=3F8D NO/BREAK=3F86
DELAY=3F87 DELAY,=3F88 DELAY,4=3F89
/KIM/GETCH=15A /KIM/DOUT=1A0 BRK,VCT,IN=3FA6
/HARDCOPY=0000 HARDCOPY/A=416B HC/RTS=4173
END/A=4174 DO/LINFEED=416B HARDCOPY/B=416A
/0/B=4174 /C/PORT=1702 IN/PORT=415B
//0000,4163,4163
The information on these sheets describe how to load the data on the supplied cassette and configure this software for your system.

The supplied cassette is in a specially recorded format which can be read by any 6502 based system with a system clock of 1 mhz. The procedure for loading the data on this cassette is as follows:

Procedure for loading the data on the cassette tape

1) Read the description of the Fast Cassette Interface. This is the software which reads the cassette data.

2) Manually enter the object code contained on the Fast Cassette Interface listing, and construct the connection to your tape deck.

3) Configure this listing per your system. The required changes pertain to cassette input/output ports and are underlined in the listing.

4) Enter the following data:

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0123</td>
<td>01</td>
</tr>
<tr>
<td>0124</td>
<td>00</td>
</tr>
<tr>
<td>0125</td>
<td>20</td>
</tr>
<tr>
<td>0126</td>
<td>FD</td>
</tr>
<tr>
<td>0127</td>
<td>3F</td>
</tr>
</tbody>
</table>

5) Insert cassette tape and position a few seconds before start of data, execute the Fast Cassette Interface software at 4141, and then press the play switch on the tape deck.

6) After approximately one minute, the data should have been loaded. If the contents of the accumulator = 00 then you have a good load. If = EE then an error was detected, and you should try again. If it appears the program "hung up", then recheck connections and the modifications to the Fast Cassette Interface software. Also try via the inverter in the circuit.

The following is not required for the following versions: KIM

Configure ASSM/TED for your system requirements

1) Configure via table A by entering the address of appropriate routines or patches.

2) If you prefer, link ASSM/TED to the Fast Cassette Interface software as follows:

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>3FA3</td>
<td>4C A5 40</td>
</tr>
<tr>
<td>3FD3</td>
<td>4C 00 40</td>
</tr>
</tbody>
</table>
C. SYM

The default file boundaries for SYM are: text file = 0200-OBFC, label file =0C00-OBFC, and relocatable buffer = 0POO. When entering the file boundary via the SET command, enter the end address minus 3 (example: If the end = OBFF, then enter OBFC).

ASSM/TED provides software for controlling two tape motors. ASSM/TED assumes the record deck (deck 0) is connected to the onboard motor control. If the user implements motor control hardware for the play deck (deck 1), ASSM/TED can control it via pin A-15 ("1" = on, "0" = off).

ASSM/TED for the SYM uses BB-F6 of zero page and most of the bottom of the stack (0100 up).
0010  ***RELOCATING LOADER FOR THE SYM-1 ASSEMBLED***
0020  
0030  
0040  
0050  .DS
0060  
0070  ======COPYRIGHT 1979 BY CARL MOSER=====
0080  ====== ALL RIGHTS RESERVED. ======
0090  
0100  
0110  
0120  
0130  ======= USER INPUTTED VARIABLES BEFORE EXECUTION =======
0140  FILE/NOD  .DE $0110  FILE NUMBER (0-99)
0150  OFFSET    .DE $E0  RELOCATOR OFFSET (2 BYTES)
0160  BUFFER    .DE $C0  ADDR. OF R.L. BUFFER
0170  
0180  
0190  
0200  
0210  
0220  DIRECTIVE  DESCRIPTION
0230  ---------  ------------
0240  0F        EXTERNAL 2 BYTE ADDR. PRECEDES,
0250  0F        DON'T RELocate. OTHERWISE RELocate.
0260  1F        .L,  DATA PRECEDES.
0270  
0280  
0290  2F        .H,  DATA PRECEDES, LD PART FOLLOWS.
0300  
0310  3F        .A,  BYTE FOLLOWS.
0320  
0330  
0340  
0350  5F        TURN RELOCATOR ON (VIA .RS).
0360  
0370  
0380  
0390  6F        TURN RELOCATOR OFF (VIA .RC).
0400  
0410  
0420  
0430  7F        .DS - 2 BYTE BLOCK VALUE FOLLOWS.
0440  
0450  
0460  .DS $0200
0470  
0480  ITAPE INPUT PARMS
0490  LOAD/NOD  .DE $0180  0: NO STORE 1: STORE
0500  TSTART    .DE $A6AC LOAD BEGINNING AT TSTART
0510  TEND      .DE $A64A STOP LOADING AT TEND
0520  
0530  
0540  HEADER INPUT DATA
0550  FILE/NOD  .DE $017A HEADER FILE NUMBER
0560 HSTART  .DE $017B HEADER START
0570 HEND    .DE $017D HEADER END
0580 1
0590 1
0600 IVARIABLES
0610 SCRT    .DE $11E SCRATCH AREA
0620 TEMP1   .DE $11F SCRATCH AREA
0630 TEMP2   .DE $120 SCRATCH AREA
0640 SAVE    .DE $121 SCRATCH AREA
0650 ADDR    .DE $DC 4 BYTES OF ADDRESS INFO.
0660 BUFF.END .DE $0123 END OF 256 BYTE BUFFER
0670 BUFF.INDEX .DE $0124 PRESENT ACCESSED DATA FROM BUFFER
0680 1
0690 1
0700 IR(P)=00: RELOCATOR ON
0710 IR(P)=02: RELOCATOR OFF
0720 1
0730 IBEGIN EXECUTION AT LABEL START
0740 1
0800- A2 FF 0750 START
0802- 9A 0760 LDY #$FF
0803- E8 0770 TXS INITIALIZE STACK
0804- DB 0780 LDY #$FF
0805- 8E 21 01 0790 STA R(X)=00
0806- 20 E3 0800 0807 JSR LOAD+BUFF
0808- 4C 11 08 0810 JMP ENTY
0809- 20 71 09 0820 LDDP1
0810 1
0820- C9 7F 0830 ENTR
0821- D0 03 0850 CMP $7F
0822- D0 03 0850 JMP PROC.3F
0823- 4C A7 09 0860 BNE proc.3F
0824- C9 3F 0870 PROC.3F
0825- D0 03 0880 JMP LOAD+BUFF
0826- C9 4F 0890 JMP LOAD+BUFF
0827- C9 4F 08A0 CMP $4F
0828- C9 4F 08B0 CMP $4F
0829- D0 03 08C0 BNE W:1
082A- 4C 00 08D0 JMP W:1
082B- C9 5F 08E0 CMP $5F
082C- D0 04 08F0 BNE CKMX
082D- A2 00 0900 LDX $00
082E- F0 D8 0910 BEO LOADP1
0830 1
0831- C9 6F 1000 CMP $6F
0832- D0 04 1010 BEO LOADP1
0833- D0 02 1020 BNE MD+REL
0834- 81 DC 1030 LDY $02
0835- 81 DC 1040 BNE LOADP1
0836- 81 DC 1050 BNE LOADP1
0837- 81 DC 1060 STA $ADD+X
0838- 81 DC 1070 JSR INC+ADDRS
0839- C9 00 1080 CMP $00
083A- 4F C7 1090 BEO LOADP1
083B- C9 20 10A0 CMP $20
083C- F0 0F 10B0 BEO LOADP1
083D- 8D 21 01 1110 STA SAVE SAVE R(A), IT CONTAINS DP CODE
083E- 29 9F 1120 AND $9F
083F- F0 BC 1130 BEO LOADP1
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0253- F 0 BC</td>
<td>1140</td>
<td>BEO LOOP1</td>
<td>0255- AD 21 01</td>
<td>1150</td>
<td>LDA SAVE RESTORE OP CODE</td>
</tr>
<tr>
<td>0256- 29 1D</td>
<td>1160</td>
<td>AND #$1D</td>
<td>0258- C9 08</td>
<td>1170</td>
<td>CMP #$0F CKG. FOR ONE BYTE INSTR.</td>
</tr>
<tr>
<td>025A- F0 B3</td>
<td>1180</td>
<td>BEO LOOP1</td>
<td>025C- C9 18</td>
<td>1190</td>
<td>CMP #$18 CKG. FOR ONE BYTE INSTR.</td>
</tr>
<tr>
<td>0260- F0 AF</td>
<td>1200</td>
<td>BEO LOOP1</td>
<td>0262- AD 21 01</td>
<td>1220</td>
<td>IGN, TEST FOR INSTR. CONTAINING 2 BYTES</td>
</tr>
<tr>
<td>0265- 29 1C</td>
<td>1240</td>
<td>LDA SAVE RESTORE OP CODE</td>
<td>0267- C9 1C</td>
<td>1260</td>
<td>AND #$1C</td>
</tr>
<tr>
<td>0269- F0 42</td>
<td>1280</td>
<td>BEO TWO+BYT+AD</td>
<td>026C- C9 18</td>
<td>1290</td>
<td>CMP #$18</td>
</tr>
<tr>
<td>026D- F0 3E</td>
<td>1300</td>
<td>BEO TWO+BYT+AD</td>
<td>026E- C9 0C</td>
<td>1310</td>
<td>CMP #$0C</td>
</tr>
<tr>
<td>0271- F0 3A</td>
<td>1320</td>
<td>BEO TWO+BYT+AD</td>
<td>0273- 20 74 03</td>
<td>1340</td>
<td>THE REMAINING INCLUDE ONE BYTE OF ADDRESS INFORMATION</td>
</tr>
<tr>
<td>0276- 01 DC</td>
<td>1350</td>
<td>JSP DATA</td>
<td>0278- 20 88 03</td>
<td>1400</td>
<td>JSP INC+ADDRS</td>
</tr>
<tr>
<td>027B- 20 74 03</td>
<td>1410</td>
<td>JSP DATA</td>
<td>027E- C0 1F</td>
<td>1440</td>
<td>CMP #$0F CKG. FOR RELLOCTR DIRECTIVE</td>
</tr>
<tr>
<td>0280- F0 14</td>
<td>1450</td>
<td>BEO IMM+HI CKG. FOR $H.</td>
<td>0282- C9 1F</td>
<td>1460</td>
<td>BNE ENTRY</td>
</tr>
<tr>
<td>0284- D0 0E</td>
<td>1470</td>
<td>JMP LOOP1</td>
<td>0286- 20 95 03</td>
<td>1480</td>
<td>JSP DATA RELCC.</td>
</tr>
<tr>
<td>0289- 18</td>
<td>1490</td>
<td>CLC</td>
<td>0292- 05 E0</td>
<td>1500</td>
<td>LDA (ADDRs+$X)</td>
</tr>
<tr>
<td>0290- 00</td>
<td>1510</td>
<td>ADD *OFFSET+$00 ADD OFFSET LOW PART FOF $L.</td>
<td>0294- 01 DC</td>
<td>1520</td>
<td>STA (ADDRs+$X)</td>
</tr>
<tr>
<td>0295- 20 88 03</td>
<td>1530</td>
<td>JSP INC+ADDRS</td>
<td>0298- 4C 11 02</td>
<td>1540</td>
<td>BACK=T0+L4 JSP LOOP1</td>
</tr>
<tr>
<td>0299- 18</td>
<td>1550</td>
<td>JMP LOOP1</td>
<td>029C- 20 74 03</td>
<td>1560</td>
<td>JSP DATA LOW BYTE FOLLOWS REL. DIR.</td>
</tr>
<tr>
<td>029F- 18</td>
<td>1570</td>
<td>CLC</td>
<td>02A2- 65 E0</td>
<td>1580</td>
<td>ADC *OFFSET FORM THE LD ADDR. PART</td>
</tr>
<tr>
<td>02A4- 08</td>
<td>1590</td>
<td>PHP</td>
<td>02A8- 20 95 03</td>
<td>1600</td>
<td>JSP DEC+ADDRS</td>
</tr>
<tr>
<td>02AA- 28</td>
<td>1610</td>
<td>PLP</td>
<td>02A1- A1 DC</td>
<td>1620</td>
<td>LDA (ADDRs+$X)</td>
</tr>
<tr>
<td>02A3- 65 E1</td>
<td>1630</td>
<td>ADC *OFFSET+$1 NOW FORM THE EFFECTIVE $H.</td>
<td>02A5- 81 DC</td>
<td>1640</td>
<td>STA (ADDRs+$X)</td>
</tr>
<tr>
<td>02A7- 20 88 03</td>
<td>1650</td>
<td>JSP INC+ADDRS</td>
<td>02AB- 4C 11 02</td>
<td>1660</td>
<td>JMP LOOP1</td>
</tr>
<tr>
<td>02AD- A0 02</td>
<td>1670</td>
<td>JMP LOOP1</td>
<td>02AF- 98</td>
<td>1700</td>
<td>PXA</td>
</tr>
<tr>
<td>02B0- 48</td>
<td>1710</td>
<td>PHA SAVE R(Y)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
0271- 80 74 03 1720 JSR GET+DATA
0274- 61 DC 1730 STA (ADDR X)
0276- 20 88 03 1740 JSR INC+ADDRS
0279- 6B 1750 PLA
0280- A8 1760 TRN RESTORE R(Y)
0281- 88 1770 DEY
0282- DF F1 1780 BNE XX
0285- 20 74 03 1790 JSR GET+DATA
0288- C9 0F 1800 CMP $0F CHK FOR RELLOCATOR DIRECTIVE
028A- 10 03 1810 BNE XY
028C- 4C 11 02 1820 JMP LOOP1
028E- 40 1830 XY PLA
0290- 20 95 03 1840 JSR DEC+ADDRS
0293- 20 95 03 1850 JSR DEC+ADDRS
0296- 80 1860 DECREMENT BACK TO ADDRESS START
1870 :
0298- A1 DC 1880 LDA (ADDR X)
029A- 18 1890 CLC
029B- 65 E0 1900 ADC *OFFSET ADD OFFSET LD
029D- 81 DC 1910 STA (ADDR X)
029F- 20 88 03 1920 JSR INC+ADDRS
02A2- A1 DC 1930 LDA (ADDR X)
02A5- 65 E1 1940 ADC *OFFSET+81 ADD OFFSET H1
02AE- 81 DC 1950 STA (ADDR X)
02AF- 20 88 03 1960 JSR INC+ADDRS
02B2- 68 1970 PLA
02B3- 4C 14 02 1980 JMP ENTY
1990 :
2000 SUBROUTINE LOAD BUFFER WITH DATA FROM TAPE
2010 :
02B6- A9 7A 2020 LOAD=BUFF LDA $10 ADDLDF OF START OF HEADER
02B8- 80 4C A6 2030 STA TSTAP+$$0
02BC- A9 7F 2040 LDA $17 ADDLDF OF END OF HEADER
02BD- 80 4A A6 2050 STA TEND+$$0
02BF- A9 01 2060 LDA $01 H1 ADDR
02C0- 80 4D A6 2070 STA TSTAP+$$1
02C4- 80 48 A6 2080 STA TEND+$$1
02C8- 80 00 01 2090 STA LOAD=NO 01: INDICATE TO LOAD
02CA- 20 DF 03 2100 JSR USER/LOAD USER LOAD=80 FROM TAPE ROUTINE
2110 :
2120 THE ABOVE SETS UP AND LOADS HEADER INFORMATION
2130 FROM TAPE. THE HEADER CONTAINS THE MODULE FILE
2140 NUMBER, AND STARTING AND ENDING ADDRESS OF FOLLOWING
2150 DATA.
2160 :
2170 :
02D6- DO 4D 2180 BNE ERROR IF Z-BIT FALSE, THEN ERROR IN LOADING
0300- A2 00 2190 LDX $00
2200 :
0302- AD 7D 01 2210 LDA HEND+$$0
0305- 38 2220 SEC
0306- ED 7B 01 2230 SBC HSTART+$$0
2240 CALCULATE NUMBER OF BYTES IN FOLLOWING DATA
2250 :
0309- 8D 23 01 2260 STA BUFF.ENW INITIALIZE BUFFER END POINTER
030C- AD 7E 01 2270 LDA HEND+$$0
030F- ED 7C 01 2280 SBC HSTART+$$0
0312- DE 39 2290 BNE ERROR ONLY 256 BYTE BUFFER ALLOWED
0314- A5 C0 2300 LDA *BUFFER
0316- 8D 4C A6 2310 STA TSTART
0319- 18 2320 CLC
031A- 6D 23 01 2330 ADC BFF,END = BYTES
031D- BD 4A 06 2340 STA TEND
0320- A5 C9 2350 LDA *BUFFER+$01
0322- 8D 4D A6 2360 STA TSTART+$01
0325- 69 00 2370 ADC $00
0327- 8D 4B A6 2380 STA TEND+$01
2390 NOTE THE START AND END ADDRESS PARMS HAVE BEEN
2400 SET UP TO LOAD FROM TAPE INTO THE BUFFER.
2410 ;
032A- AD 10 01 2420 LDA FILE=NO USER ENTERED FILE NUMBER
032D- F0 08 2430 BEO STORE,DATA IF F= 00, LOAD ANYWAY
0330- CD 7A 01 2440 CMP MFILE=NO CMP WITH USER VERSUS THAT ON TAPE
0332- F0 05 2450 BEO STORE,DATA
0334- BE 80 01 2460 STX LOAD=NO R(X)= 01 NO STORE
0337- 20 D5 05 2470 STORE,DATA JSR USER=LOAD
2480 ;
2490 THE ABOVE LOADS INTO DATA INTO BUFFER DEPENDING
2500 ON THE STATE OF LOAD=NO
2510 ;
033A- D0 11 2520 BNE ERROR Z-BIT = FALSE THEN ERROR
033C- AE 00 2530 LDX #800
033E- AD 7A 01 2540 LDA MFILE=NO
0341- C9 EE 2550 CMP #800 COMPARE IF END OF FILE
0343- D0 0C 2560 BNE BUFFERED
0345- A9 00 2570 LDA #800 INDICATE GOOD LOAD
0347- 00 2580 B BRK
0348- EA 2590 NOP
0349- EA 2600 NOP
034A- 4C 00 02 2610 JMP START
034D- A9 EE 2620 ERROR LDA #800 INDICATE ERROR IN LOAD
034F- D0 F6 2630 BNE #
2640 ;
2660 INTH THE ADDR., INFO., AND PUT IN ADDR+$2, +$3
2670 ADDR INFO. IS IN FIRST TWO BYTES OF BUFFER
2680 ;
0351- AD 80 01 2690 BUFFERED LDA LOAD=NO CKG. IF PROPER DATA
0354- F0 90 2700 BEO LOAD=BUFFER
0356- AE 21 01 2710 LDX SAVE=RESTORE R(X)
0359- A0 00 2720 LDY #800
035B- B1 C8 2730 LDA <BUFFER>,Y
035D- 85 DE 2740 STA *ADDRS+$2
035F- C8 2750 INY
0360- B1 C8 2760 LDA <BUFFER>,Y
0362- 85 DF 2770 STA *ADDRS+$3
0364- 8C 24 01 2780 STY BUFF,INDEX SET BUFFER DATA POINTER
2790 ;
2800 SET RELLOCATION ADDR., IN ADDR+$0, +$1
0367- A5 DE 2810 LDA *ADDRS+$2
0369- 10 2820 CLC
036A- 65 E0 2830 ADC #OFFSET
036C- 85 DC 2840 STA *ADDRS
036E- A5 E1 2850 LDA *ADDRS+$1
0370- 65 DF 2860 ADC *ADDRS+$3
0372- 85 DD 2870 STA *ADDRS+$1
PAGE 07

03CA- 85 DE 3460 STA @ADDRS+2
03CC- 98 3470 TYA @GET HI
03CD- 85 DF 3480 ADC @ADDRS+3
03CF- 85 DF 3490 STA @ADDRS+3
03D1- 68 3500 ND.PDC PLA
03D2- 4C 11 02 3510 JMP LDDP1
3520 ;
3530 ;
3540 ;
3550 ;
3560 ;
3570 ;
3580 ***SYM CASSETTE INTERFACE PATCH ***
3590 ;
3590 SYM DEFINITIONS:
3600 SAVER .DE $9168
3600 ACCESS .DE $8E86
3610 ID .DE $A64E
3620 MODE .DE $FD
3630 CONFIG .DE $89A5
3640 ZERCK .DE $832E
3650 P2SCR .DE $829C
3660 LOADT .DE $8C78
3670 NACCESS .DE $808C
3680 RESXAF .DE $81B8
3690 ;
3700 ;
3710 USER=LOAD JSR SAVER ;SAVE REGISTERS
03D2- 86 81 3720 LDA .FF ;ID=FF FOR USER RANGE
03DA- 8D 4E A6 3730 STA ID
03DD- 88 80 3740 LDR .#$00
03DF- 84 FD 3750 STY @MODE ;BIT 7=1 FOR H.S.
03E1- A9 09 3760 LDA .#$09
03E3- 20 A5 89 3770 JSR CONFIG
03E6- 20 8F 83 3780 JSR ZERCK
03E9- 20 9C 82 3790 JSR P2SCR
03EC- 20 7E 8C 3800 JSR LOADT+$3 ;ENTRY IN TAPE LOAD
03EF- D8 3810 CLD
03F0- A9 00 3820 LDA .#$= ;12-BIT =T
03F2- 90 02 3830 BCC SKPERPL/L
03F4- A9 01 3840 LDA .#$=1 ;12-BIT =F
03F6- 4C 88 81 3850 JMP RESXAF ;RESTORE REGS. EXCEPT A+PSR
3870 ;
3880 ;
3890 ENDPGM .EN

LABEL FILE: [ / = EXTERNAL ]

/FILE/NO=0110 /OFFSET=00E0 /BUFFER=00C8
/LOAD/NO=0180 /TSTART=A64C /TEND=A64A
/HFILE/NO=017A /HSTART=017B /HEND=017D
/SCRAP=011E /TEMPl=011F /TEND=0120
/SAVE=01d1 /ADDRs=00DC /BUFF.END=0123
/BUFF.INDEX=0124 START=0200 /LODDP1=0211
ENTRY=0214 PRO.3F=0218
M=0231 CKNX=0239
ONE+BYT+AD=0273 IMM+LO=0286
BACK+TD+L1=0293


IMM+HI=0296
XY=02C8
B=0347
GET+DATA=0374
SKIP+INC1=038E
SKIP+DEC1=039F
PROD.IS=0389
/ACCESS=8B86
/CONFIG=8F95
/LOADT=8C78
USER/LOAD=03D5
</0000,03F9,03F9>
TWO+BYTE+AD=02AD
LOAD+BUFF=02E6
ERROR=034D
WX=0385
/ACCESS=8B86
/CONFIG=8F95
/LOADT=8C78
USER/LOAD=03D5
XX=02AF
STORE.DATa=0337
BUFFLOADED=0351
INC+ADDR=0388
DEC+ADDR=0395
PRD.7F=039A
/SAVER=8188
/MDME=00FD
/P2SCR=829C
/RESXAF=8188
END.PGM=03F9