Synertek
Systems
Corporation

RAE-1
REFERENCE MANUAL

RAE-1 REFERENCE MANUAL

SSC Pub MAN-A-260027-C

Third Printing: October, 1980

Synertek Systems Corporation

P.O. BOX 552 SANTA CLARA, CALIFORNIA 95052 TEL. (408) 988-5689 TWX 910-338-0135

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT)

LIST OF TABLES

APPENDICES

SECTION 1.0

INTRODUCTION TO RAE

This 6502 resident relocating macro assembler and text editor reside simultaneously in 8K bytes of ROM memory. Sufficient memory must be provided 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 it also. On cold start entry, the RAE will set the file boundaries as follows.

- Text file = 0200-0BFC
- Label file = 0C00-0EFC
- Relocatable object buffer = 0F00

The label file and text file that RAE generates are position independent and may be located practically anywhere in RAM memory. The object code file location is dependent on the beginning of assembly (.BA) and the move code (.MC) pseudo ops.

RAE 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 efficient utilization of memory.

Some major features of RAE are:

- Macro and conditional assembly support
- Labels up to 10 characters in length
- Auto line numbering for ease of text entry.
- Creates either executable code in memory or relocatable object code on tape
- Manuscript feature for composing letters and other text
- Loading and storing of text on tape
- Supports up to two tape decks, terminal with keyboard, and printer
- String search and replace capability, plus other powerful editing commands
- Upper and lower case accepted

Throughout this document, output generated by RAE is underlined if necessary to distinguish it from user input.

Initial entry (cold start) to RAE is at address B000. Warm start is at address B003. If the break command (≥BR) is executed, one may return to the address following the break. Initial entry provides the following default parameters:

FOR TED

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

FOR ASSEMBLER

- Assumes assembling from memory (otherwise use .CT)
- Does not store object code in memory (otherwise use .OS)
- Begins assembly at $0200 (otherwise use .BA)
- Output listing clear (otherwise use .LS or ≥ASSEMBLE LIST)
- Stops assembly on errors (otherwise use .CE)
- Stores object code beginning at $0200 unless a .BA or .MC is encountered and if .OS is present
- Generates relocatable addresses
- Macro object code is not output (otherwise use .ES)

The RAE is designed to operate with a cassette record unit and a play unit. A single record/play unit may be used but one will not be able to create relocatable object files when assembling from tape.

When inputting to RAE the following control codes are useful:

CONTROL H (hex 08), Rubout or Delete (hex 7F)	Backspaces over previous character. More than one of these may be entered to delete a number of characters. A backslash is echoed if rubout is depressed.
CONTROL X (hex 18)	Deletes the entire line.
Break Key	Halts outputting, and waits for input of appropriate control code.

For a more detailed list see Section 8.

SECTION 2.0

GETTING STARTED WITH RAE

2.1 GENERAL

An assembler is a program which allows the user to compose and enter programs at the machine language level in a form that is much more convenient than actual machine code. The assembler accepts mnemonic names for individual instructions, allows symbolic names to be assigned to memory locations and data, provides for address arithmetic in terms of symbolic names, and certain other features, depending on the sophistication of the assembler in question.

The Synertek System's Resident Assembler/Editor (RAE) is a full features assembler. Other major features include: macros with nesting capability, conditional assembly, creation of relocatable object code supported by a relocating loader, string search/replace and line editing, automatic control of two I/O tape units, and assemble directly from tape.

It is commonly thought that the primary feature offered by an assembler is that of writing machine instructions in a more convenient form. However, this is only one aspect of the advantage of an assembler, and perhaps not even the most significant. The use of symbolic names to represent numbers makes variables of what most likely would have been considered constants. The very presence of symbols bestows a generality and flexibility to a program which otherwise might have seemed quite rigid. This encourages the programmer to abstract the immediate problem and perhaps develop a more adaptable program. Also, since the actual calculation or assignment of a value to a symbol can be deferred, the development of logically separate modules can proceed freely. Programs so organized become much more readable and managable, both in their maintenance and amenability to revision.

The purpose of an assembler is to translate a program written in assembly language into machine language. Machine language refers to that representation of instructions which are immediately interpretable by the machine being considered. For all intents and purposes, the machine language of the 6502 consists of hexadecimal opcodes and data. An assembly language is a symbolic representation of machine language instructions; e.g., LDA # is used to represent the instruction A9, LoaD the Accumulator with the value following the # sign.

The program written in assembly language is called the source code, the machine language program produced by the assembler is called the object code.

With or without an assembler, it should be realized that programs are usually written in assembly language. The assembler simply saves us the tedious and error-prone task of translating our program into machine code.

The assembler accomplishes the conversion of the source code to machine code in two passes; that is, the source program is scanned twice. During the first pass all symbols and their associated values are collected into a label file (also called a symbol table). During the second pass the assembler converts the program to machine language (also called object code), using the definitions collected in the first pass.

One important feature that all assemblers share is that of assembler directives, or pseudo ops. These are special orders to the assembler itself about the way it is to deal with the source program, or for the definition and manipulation of symbols and allocation of storage. The distinction between operations (machine instructions) and psuedo operations is similar to that between a manuscript to be typed and the author's marginal notes to the typist. For example, directives are used to tell the assembler to set aside 100 memory locations to be used for an array, or to tell it where the object code is to be stored in memory.

2.2 HARDWARE PREPARATION

2.2.1 RAE ROM ADDRESSING

Your RAE is contained in one (RAE-1) or two (RAE-1/2) ROMs. These ROMs are designed to run under the SYM-1 SUPERMON monitor or the MDT-1000 system monitor.

Before you install the ROMs into your system, refer to your system reference manual to locate or "strap" the desired ROM socket at the correct memory address as shown below.

RAE-1	RAE-1/2
● Single 8K byte chip (2364)	● Two 4K byte chips (2332's)
● P/N 02-0053A	● P/N's 02-0023A and 02-0024A
● Chip select pin 20	● Chip select pin 20
● Address B000-BFFF, E000-EFFF	● Address B000-BFFF (02-0023A) E000-EFFF (02-0024A)

For detailed discussion of jumper configurations for SYM-1, see Section 10.0, paragraph 4.

2.2.2 AUDIO CASSETTE I/O

RAE is designed to work with a dual audio cassette system using Synertek Systems' high speed recording format. Cassette unit #0 is designated the record unit and unit #1 is designated the play unit. A single cassette player may be used for most operations except where the user wishes to assemble source from tape and store object code back onto tape.

Refer to your particular system's reference manual for details on I/O addressing, remote control and adjustments. The following is a summary of each of these:

ADDRESSING (IN, OUT, REMOTE CONTROL)

SYSTEM	AUDIO IN	AUDIO OUT	REMOTE CONTROL (RECORD) # 0	REMOTE CONTROL (PLAY) # 1
SYM-1	A000-BIT 6	A400 (SYM ref. manual)	A00C-CB2	A000-BIT 7
MDT1000	9600-BIT 7	9600-BIT 6	9703-CB2	9701-CA2

AUDIO CASSETTE RECORDER ADJUSTMENTS

TONE	High or treble
VOLUME*	2V peak-to-peak or saturation (max volume) from recorder (suggested for most recorders)
TAPE	Data tape or high quality, low noise audio tape Short lengths (30 min or less) works best
SUGGESTED RECORDER	Sanyo M2544A or equivalent

*** Each recorder type will require a volume adjustment in order to obtain maximum reliability, 2vp-p or saturation (max volume) works well on most recorders.**

2.3 STEP-BY-STEP EXAMPLE

To access the Resident Assembler/Editor (RAE), power up your system and log-on to your terminal, then type "G B000"; this is the cold start entry point. (The warm start entry point is B003).

RAE will respond with:

```
RAE V1.0
COPYRIGHT 1979 SYNERTEK SYSTEMS CORP.


0200-0BFC  0C00-0EFC  0F00
0200  0C00
```

NOTE: **TEXT FILE 0200-0BFC**
LABEL FILE 0C00-0EFC
RELOCATABLE OBJECT BUFFER 0F00
CURRENT END OF TEXT BUFFER 0200
CURRENT END OF LABEL BUFFER 0C00

If you inadvertently stop RAE log-on printout before the first prompt character (>) is displayed, RAE will double echo each character typed and also ignore any commands. To exit this mode, type CONTROL O.

The ">" is the prompt symbol from RAE, indicating it is ready to accept commands. In the following procedures the ">" is not shown. Only the most commonly used commands and the major features of RAE will be discussed in the following section. Several examples will be used to illustrate their use and action.

NOTE

ALL COMMANDS MUST BE ENDED WITH A CARRIAGE RETURN. If you make a typing error, enter a CONTROL H or a RUBOUT to delete the last character. Several CONTROL H's can be entered to remove more than one character. A CONTROL X will eliminate the entire line. Processing can be suspended by pressing the BREAK key and resumed with a CONTROL Q.

We will begin by entering a program segment which fills page 3 (0300-03FF) of memory with zeros. Each line of text must be preceeded by a line number, so that RAE can order them properly, as well as process any changes we may wish to make as we go along.

Type in the following lines exactly as they appear, immediately following the prompt symbol:

```
10  LDX #0
20  TXA
30LOOP STA $300,X
40  DEX
50  BNE LOOP
```

Note that the instruction mnemonics and addressing mode formats are those defined and described in the SY 6500 Programming Manual (MNA-2).

Now type in:

```
PRINT
```

RAE will respond with:

```
0010              LDX #0
0020              TXA
0030   LOOP       STA $300,X
0040              DEX
0050              BNE LOOP
//
```

Notice that RAE automatically lines up the label, instruction, and operand fields, and that if the first character is a blank the label field is skipped.

To examine lines 20 through 40 only, type in:

```
PRINT 20 40
```

RAE will reply with:

```
0020            TXA
0030  LOOP      STA $300,X
0040            DEX
//
```

Notice that the line numbers in the PRINT command are separated by blanks, not commas. This is the convention used by RAE in specifying all command parameters.

Let us now try to assemble our program.

Type in:

```
ASSEMBLE LIST
```

RAE will print:

```
                  0050              BNE LOOP
!07 AT LINE 0050/44
```

This is an error message, telling us that the .EN (end of program) pseudo op is missing. It is required to indicate to RAE the end of the source program. Let us put it in and try again.

Type in the following:

```
60 .EN
ASSEMBLE LIST
```

RAE will respond with:

```
0200- A2 00    0010               LDX #0
0202- 8A       0020               TXA
0203- 9D 00 03 0030 LOOP          STA $300,X
0206- CA       0040               DEX
0207- D0 FA    0050               BNE LOOP
               0060               .EN

LABEL FILE:  [ / = EXTERNAL ]

LOOP=0203
//0000,0209,0209
```

This time assembly of our program was successful. The listing produced shows us the object code as well as the source code. The leftmost column contains the address of the first byte of each instruction. As can be seen, the default beginning address is $200. The .BA (begin assembly) pseudo op is used when we wish RAE to assemble beginning at some other address, say $500.

Type in:

```
5 .BA $500
ASSEMBLE LIST
```

RAE will respond with:

```
                       0005                  .BA $500
0500- A2 00            0010                  LDX #0
0502- 8A               0020                  TXA
0503- 9D 00 03         0030 LOOP             STA $300,X
0506- CA               0040                  DEX
0507- D0 FA            0050                  BNE LOOP
                       0060                  .EN

LABEL FILE:  [ / = EXTERNAL ]

LOOP=0503
//0000,0509,0509
```

Up to this point everything RAE has done has been "on paper". If we want the object code generated by RAE to be actually stored in memory at the address specified, we need to include the .OS (object store) pseudo op. Type in the following:

```
6 .OS
ASSEMBLE
```

Notice that the LIST option was omitted from the ASSEMBLE command.

This time RAE will simply print:

```
//0000,0509,0509
```

Let us exit RAE momentarily to examine some memory locations. To exit to the system monitor type:

```
BREAK                (or CONTROL C)
```

The system monitor will print:

```
B0AC,0
```

Now type:

```
V 0500
```

The system monitor will reply:

```
0500 A2 00 8A 9D 00 03 CA D0,66
 0366
```

This is the object code of our program, stored by RAE. To continue where we left off type either:

```
G B003   or G
```

B003 is the warm start entry point to RAE. If the cold entry point were used our text file would be lost.

RAE will print:

```
0200-0BFC  0C00-0EFC  0F00
0248  0C06
```

In order to execute our program without exiting RAE, we need to make the last executable instruction an RTS so that control will be returned to RAE.

Type in:

```
55 RTS
ASSEMBLE
```

RAE will print:

```
//0000,050A,050A
```

Now enter:

```
RUN $500
```

RAE will come back with the prompt sign, ">". Let us exit RAE again to verify that the program ran.

Type:

```
BREAK
```

System monitor will print:

```
B0AC,0
.
```

Now type:

```
V 0300
```

System monitor will print:

```
0300 00 00 00 00 00 00 00 00,00.
 0000
.
```

Apparently our program worked as intended. Get back into RAE. Recall that the warm entry point is B003.

Let us begin a new example. This time we will change the starting boundary of the text file to allow room for object code to be stored in memory at the RAE default origin. Type the following:

```
SET $300
```

RAE will respond with:

```
0300-0BFC  0C00-0EFC  0F00
024E  0C06
```

We must now clear the text file because its starting boundary has been changed. Failure to do so is catastrophic. To do this type:

```
CLEAR
```

If you now type PRINT, RAE will simply print //, which is the end-of-text indicator. The following code is for a pseudo-random number generator. To make the entering of the text easier, first type in:

```
AUTO 10
```

This command enables the automatic line numbering option. The 10 will be used as the line number increment. AUTO goes into effect after a line is referenced.

Type in:

```
100RND SEC
```

RAE will now respond with:

```
0110>
```

which is the next line number. Now enter the following lines after each line number, remembering to leave a space if there is no label:

```
 LDA TABLE+1
 ADC TABLE+4
 ADC TABLE+5
 STA TABLE
 LDX #4
MOVE LDA TABLE,X
 STA TABLE+1,X
 DEX
 BPL MOVE
 RTS
```

To exit AUTO, type:

```
//
```

We must be sure to include the .EN (end of program) pseudo op, so enter:

```
999 .EN
```

RAE will respond with:

```
1009>
```

This is because the AUTO mode is still enabled. Type // to exit AUTO, then, to turn off the AUTO option, type:

```
      AUTO 0
```

Let's now try to assemble our code. Enter:

```
      ASSEMBLE LIST
```

RAE will reply:

```
0200- 38            0100 RND          SEC
                    0110              LDA TABLE+1
!08 AT LINE 0110/00
```

This error message tells us that there is an undefined label in line 110. The problem is, of course, that RAE has no way of knowing what the symbol TABLE represents. TABLE is meant to be the name of an array of six elements. The pseudo op .DS (define storage) is used to tell RAE to set aside a specified number of memory locations.

Type in:

```
      90TABLE .DS 6
      ASSEMBLE LIST
```

RAE will print:

```
0200-               0090 TABLE        .DS 6
0206- 38            0100 RND          SEC
0207- AD 01 02      0110              LDA TABLE+1
020A- 6D 04 02      0120              ADC TABLE+4
020D- 6D 05 02      0130              ADC TABLE+5
0210- 8D 00 02      0140              STA TABLE
0213- A2 04         0150              LDX #4
0215- BD 00 02      0160 MOVE         LDA TABLE,X
0218- 9D 01 02      0170              STA TABLE+1,X
021B- CA            0180              DEX
021C- 10 F7         0190              BPL MOVE
021E- 60            0200              RTS
                    0999              .EN
LABEL FILE:   [ / = EXTERNAL ]

TABLE=0200                   RND=0206                   MOVE=0215

//0000,021F,021F
```

Notice that TABLE has been assigned the address 200 (hex), and that the first byte of code is at location 206. Thus locations 200 - 205 have been reserved; TABLE+1 is memory location 201, TABLE+2 is 202, etc.

To test this routine we will add some code which will call RND as a subroutine and print out the pseudo random numbers generated. To aid us in the output we will call on two subroutines in the system monitor: OUTBYT and CRLF. OUTBYT outputs the contents of the accumulator as two hex digits, and CRLF outputs a carriage return and a line feed.

In order to use them, we must tell RAE where they are located. This is done using the .DE (define external) pseudo op, which tells RAE that the addresses specified are external to our program. Type in the following lines:

```
40 .OS
50OUTBYT .DE $82FA
60CRLF .DE $834D
300START LDY #8
310NEXT JSR RND
320 LDA TABLE
330 JSR OUTBYT
340 JSR CRLF
350 DEY
360 BNE NEXT
370 RTS
```

Assemble, and check that your output looks exactly as follows:

```
                   0040              .OS
                   0050 OUTBYT       .DE $82FA
                   0060 CRLF         .DE $834D
0200-              0090 TABLE        .DS 6
0206- 38           0100 RND          SEC
0207- AD 01 02     0110              LDA TABLE+1
020A- 6D 04 02     0120              ADC TABLE+4
020D- 6D 05 02     0130              ADC TABLE+5
0210- 8D 00 02     0140              STA TABLE
0213- A2 04        0150              LDX #4
0215- BD 00 02     0160 MOVE         LDA TABLE,X
0218- 9D 01 02     0170              STA TABLE+1,X
021B- CA           0180              DEX
021C- 10 F7        0190              BPL MOVE
021E- 60           0200              RTS
021F- A0 08        0300 START        LDY #8
0221- 20 06 02     0310 NEXT         JSR RND
0224- AD 00 02     0320              LDA TABLE
0227- 20 FA 82     0330              JSR OUTBYT
022A- 20 4D 83     0340              JSR CRLF
022D- 88           0350              DEY
022E- D0 F1        0360              BNE NEXT
0230- 60           0370              RTS
                   0999              .EN

LABEL FILE:  [ / = EXTERNAL ]


/OUTBYT=82FA                 /CRLF=834D                  TABLE=0200
RND=0206                     MOVE=0215                   START=021F
NEXT=0221
//0000,0231,0231
```

Since the .OS (object store) pseudo op was present, the object code was stored in memory, so we can now run the program. Type in:

```
RUN START
```

The output you get will depend on what values happened to be in memory at locations 200-205. With 20 (hex) in each location, the output will be:

```
61
A2
E3
25
A7
AC
33
3C
```

It is common practice to place all subroutines after the main body of the program. Thus, in the above example, we would like to place lines 100 through 200 after line 370. The MOVE command allows this to be done very easily. Type in:

```
MOVE 370 100 200
```

To see what has been done, enter:

```
PRINT 360 999
```

RAE will print:

```
0360           BNE NEXT
0370           RTS
0370  RND      SEC
0370           LDA TABLE+1
0370           ADC TABLE+4
0370           ADC TABLE+5
0370           STA TABLE
0370           LDX #4
0370  MOVE     LDA TABLE,X
0370           STA TABLE+1,X
0370           DEX
0370           BPL MOVE
0370           RTS
0999           .EN
```

If you type PRINT 100 200 you will see that lines 100 through 200 no longer exist. Since all the moved lines have been given the same number, we would like to renumber the text file. That is the purpose of the NUMBER command.

Type in:

```
NUMBER 90 10
```

The 90 specifies the line to begin the renumbering, and the 10 specifies the increment to use. If you now PRINT out the entire file you will see that each line number is again unique.

NOTE

The following example will utilize the audio cassette storage unit. If your cassette unit is not connected or adjusted refer to your system reference manual.

The next example is a routine which multiplies the contents of memory location MLTPLR times the contents of location MLTPND. The product will be two bytes long; the high part will be in the accumulator and the low part in location RESLO. OUTBYT will again be used to output the result. Type in:

```
CLEAR
AUTO 10
100MULT LDA #0
```

RAE will respond with:

```
0110>
```

Now enter the following lines after each line number:

```
 STA RESLO
 LDX #8
LOOP LSR MLTPLR
 BCC NOADD
 CLC
 ADC MLTPND
NOADD LSR A
 ROR RESLO
 DEX
 BNE LOOP
;LINE 210
 JSR OUTBYT
 LDA RESLO
 JSR OUTBYT
 RTS
 .EN
 //
 AUTO 0
```

Line 210 is a comment line. A comment line begins with a semicolon and may contain any characters after that, as comment lines are ignored by RAE. In this case, it is used to separate the multiplication routine from the output section for better readability. Comments may also appear on any text line by simply separating the text and comment by at least one space. As an example, retype lines 100 and 110 as follows:

```
100MULT LDA #0  ZERO RESULT HI
110 STA RESLO  ZERO RESULT LOW
```

Before this routine will assemble we need to define the symbols OUTBYT, RESLO, MLTPLR and MLTPND. Type in:

```
40OUTBYT .DE $82FA
50RESLO .DS 1
60MLTPLR .BY 2
70MLTPND .BY 3
```

The .BY (store bytes of data) pseudo op directs RAE to store the following value in the next memory location. MLTPLR and MLTPND will thus contain the numbers 2 and 3, respectively.

Finally, we need to add the .OS (object store) pseudo op, and let us also put in the .LS (print source listing on pass 2) pseudo op which enables the list option on assembly. Enter:

```
10 .OS
20 .LS
ASSEMBLE
```

RAE will print:

```
                    0010              .OS
                    0020              .LS
                    0040 OUTBYT       .DE  $82FA
0200-               0050 RESLO        .DS  1
0201- 02            0060 MLTPLR       .BY  2
0202- 03            0070 MLTPND       .BY  3
0203- A9 00         0100 MULT         LDA  #0 ZERO RESULT HI
0205- 8D 00 02      0110              STA  RESLO ZERO RESULT LOW
0208- A2 08         0120              LDX  #8
020A- 4E 01 02      0130 LOOP         LSR  MLTPLR
020D- 90 04         0140              BCC  NOADD
020F- 18            0150              CLC
0210- 6D 02 02      0160              ADC  MLTPND
0213- 4A            0170 NOADD        LSR  A
0214- 6E 00 02      0180              ROR  RESLO
0217- CA            0190              DEX
0218- D0 F0         0200              BNE  LOOP
                    0210 ;LINE 210
021A- 20 FA 82      0220              JSR  OUTBYT
021D- AD 00 02      0230              LDA  RESLO
0220- 20 FA 82      0240              JSR  OUTBYT
0223- 60            0250              RTS
                    0260              .EN

LABEL FILE:  [ / = EXTERNAL ]


/OUTBYT=82FA              RESLO=0200              MLTPLR=0201
MLTPND=0202               MULT=0203               LOOP=020A
NOADD=0213
//0000,0224,0224
```

If your output looks exactly as the above, the program is ready to be run.

Type in:

```
RUN MULT
```

The output will be:

```
0006
```

Now change the values in lines 60 and 70, assemble the new program and run it. For example the product of 4 and 9 is 0024 (hex), and that of 45 and 68 is 0BF4 (hex).

One of the most important and fundamental features of RAE is the ability to read and write to the cassette unit. We will save on tape and then retrieve the current program. Place a blank tape in your recorder, advance tape beyond blank leader and put the recorder in record mode. Now type:

```
PUT F1
```

After the file has been recorded RAE will return with the prompt. Repeat this procedure twice more to ensure a good recording. We will now read in the text file just recorded. Rewind the tape.

Now put the tape unit in the play mode, and type in:

```
GET F1
```

When the file has been read in successfully, RAE will print:

```
F01   011F  0200-031F
```

If you now type PRINT, you can verify that the file was read in correctly.

If an error occurs, retype GET F1 and start the tape again.

Now that you are acquainted with the basic features offered by RAE, you are encouraged to read Sections 3 and 4 in order to become familiar with the many other commands, pseudo ops, and editing features available to you. By far the most effective, efficient and enjoyable way to do this is to construct examples to try out each feature. Learning by doing will show you exactly how each feature works, and will enable you to utilize the full potential of the Synertek System's Resident Assembler/Editor.

SECTION 3.0

TEXT EDITOR (TED)

3.1 TEXT EDITOR COMMANDS

The TED provides 27 command functions. 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 ASM/TED considers only 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. For alphabetic parameters, only the first character is considered. For example "FORMAT CLEAR" is the same as "FO C."

NAME	EXAMPLE	PURPOSE/USE
≥ASSEMBLE w x	≥AS LI ≥AS N ≥AS L 200	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 not entered then an errors only output will be generated.
≥AUTO x	≥AU 10 ≥AU ≥AUTO 20	Automatic line numbering occurs when an x value not equal to zero is entered. x specifies the increment to be added to each line number. Auto line numbering starts after entering the first line. To prevent auto line numbering from reoccurring, enter ≥AU or ≥AU 0, after first exiting with //.
≥BREAK	≥BR ≥BRK	Break to system monitor (executes BRK instruction). A return to the TED can be performed at the address immediately after the break instruction, has the same effect as CONTROL C.
≥CLEAR	≥CL	Clear text file and turn off tape units.

NAME	EXAMPLE	PURPOSE/USE
≥COPY x y z	≥CO 110 10 40 ≥CO 300 100 200	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.
≥DELETE x y	≥DE 40 ≥DE 100 301	Delete entries in text file between line numbers x and y inclusive. If only x is entered, only the first occurence of that line is deleted.
≥DUPLICATE Fw	≥DUP ≥DUP F10 ≥DU F	Duplicate files from tape unit 1 to tape unit 0 until file w. This command starts by reading the next file on tape 1 and if that file is file w or an end of file mark then it stops. If not, the file just read will be written to tape 0 and then tape 1 is read again. This continues until file w or an end of file record is encountered.
≥FORMAT w	≥FO S ≥FO C ≥FO SET ≥FORMAT S	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.
≥GET Fx y	≥GE ≥GET F13 100 ≥GET APPEND ≥GET F2 A	Get text file with data associated with file number x from tape. The data will be loaded at line number y, or will be appended to end of the text file if the key-word APPEND is entered for y. Defaults are x = 00 and y = 0.

NAME	EXAMPLE	PURPOSE/USE
≥HARD w x	≥HA S 1 ≥HARD C ≥HA P	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 ≥HA PAGE results in the printer advancing to the top of the next page. ≥HA set will cause output to go through the printer vector in addition to OUTVEC.
≥LABELS	≥LA ≥ LAB	Print out the label file generated by the previous ASSEMBLE.
≥MANUSCRIPT w	≥MA S ≥MA C	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 RAE, 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, e.g., 'TAB' (CONTROL I).
≥MOVE x y z	≥MO 110 10 40 ≥MO 300 100 200	Move lines y thru z in the text file to just after line number x. The moved lines will all have line numbers equal to x. The original lines y thru z are deleted.
≥n	≥10 ≥100	Any entry beginning with one or more decimal digits is considered an entry/deletion of text. See Section 3.4.
≥nnnn//	≥2000//	Used to exit temporarily from auto line number mode so that commands may be entered. Entry of a line number rather than a command will cause return to auto line number mode.

NAME	EXAMPLE	PURPOSE/USE
>NUMBER x y	>NU 0 10 >NU 100 10	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.
>OFF n	>OF 0 >OF 1 >OFF	Turn off tape unit n, where n is 0 (record unit), or 1 (play unit). If an n is not entered, 0 is assumed.
>ON n	>ON 0 >ON 1 >ON	Turn on tape unit n, where n is 0 (record unit), or 1 (play unit). If an n is not entered, 0 is assumed.
>OUTPUT Fw	>OU F >OU F14 >OUT	Create a relocatable object file on tape unit 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 relocated via the >SET command.
>PASS	>PA >PASS	Execute the second pass of assembly. Not required if source is all in internal memory and the .CT pseudo op is not encountered.
>PRINT x y	>PR >PRINT 10 >PRINT 100 301	Print the text file data between line number x and y on the terminal. If only x is entered, only that line is printed. If no x and y, the entire file is outputted.
>PUT Fw x y	>PU F13 >PU F13 200 300 >PUT F >PUT	Put text file between lines x and y inclusive to tape, 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 >PU X an end of file mark is recorded.

NAME	EXAMPLE	PURPOSE/USE
≥RUN label expression		
	≥RU START ≥RU $1000 ≥RUN TEST+5	Run (execute) a previously assembled program. If a symbolic label is entered, the label file is searched for its value. The called program should contain a JMP warm start (4C03B0) as the last executable instruction.
≥SET ts te ls le bs		
	≥SE ≥SE $1000 $2000 $200 $3FF $400 ≥SET	
		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. 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). Parameters may be entered either in decimal form, or if preceded by a $, in hex form.
≥USER	≥US ≥USR	User defined command. The RAE will transfer control to location $0003. The user routine can re-enter RAE via a JMP warm start (4C03B0).

3.2 EDIT AND FIND COMMANDS

STRING SEARCH AND REPLACE (EDIT) COMMAND

≥EDIT string or ≥EDIT n

A powerful string search and replace, and line edit capability is provided via the ≥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

```
                 #
                 *
≥EDIT tS1tS2t %d Δ x y
```

where:

t	is any non-numeric terminator, e.g., ".", "/"
S1	is string to search for
S2	is string to replace S1
d	is "don't care" character. Precede with % character to change the don't care; this character used within S1 indicates which position to ignore for a search "match" condition
*	indicates to interact with user via subcommands before replacing S1 (see below)
Δ	(a space character) indicates to alter and print all lines altered
#	indicates to alter but provide no 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 this line - don't alter
X	exit ≥ED command
CONTROL F	enter form 2

Default:

d =	%
x =	0
y =	9999
Δ =	(space) print all lines altered

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

```
≥EDIT /LOOP/START/ 100 600
```

To simply delete all occurences of LOOP, enter:

```
≥EDIT /LOOP// 100 600
```

Use the * and # as described.

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

At the end of the ≥EDIT operation, the number of occurences of the string will be output as //xxxx where xxxx is a decimal quantity.

FORM 2

```
≥EDIT n
```

where:

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

Subcommands:

CONTROL F	Find user specified character
CR	(carriage return) - Retain any remaining part of a line
CONTROL D	Delete any remaining part of line
CONTROL H	Delete a character

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

```
LOOP1   LDA   #L,CRTBUFFER   LOAD FROM BUFFER
```

type CONTROL F followed with A, then CONTROL H, then Y, and then terminate line with a carriage return.

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

FIND STRING S1 COMMAND

Used to find certain occurences of a particular string. It's form is:

```
                #
                *
≥FIND tS1t %d Δ x y
```

where:

t, S1, %, d, x, y are as defined in the EDIT command, FORM 1.

*, Δ indicates to print all lines containing occurences of S1

indicates no printout

At the end of the ≥FIND operation, the number of occurences 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  /%/#
```

3.3 HOW TO USE EDIT AND FIND

We will show with a simple example, how to use some of the EDIT features of RAE. Other features, such as the use of a "don't care" character in string searching, and the control of the degree of user interaction, are described elsewhere in this manual. FIND is used to search for, but not alter, strings. It is particularly useful in finding cross-references in a source code; its use is like that of the form of EDIT which does not use a line number.

Let the text to be edited be manuscript, rather than source code. SET FORMAT CLEAR, AUTO 10, and enter the manuscript. After entry, print and examine, and make the desired corrections.

For example, let the manuscript read:

"10 Now is the time for all good men"

and let it be corrected to read:

"10 Now is the best time for most good women"

The procedure is as follows:

```
 pr
 0010  Now is the time for all good men
//

>ed 10
Now is the time for all good men
↑F>eNow is the best↑F>l time for al↑F>ll\\\most
  0010  Now is the best time for most good men
>ed /men/women/*
 23  0010  Now is the best time for most good men
*a
  0010  Now is the best time for most good women
//0001
>pr
 0010  Now is the best time for most good women
//

>
```

All underlined characters and symbols are RAE outputs.

For insertions, find the starting point and enter a new material, ending with RETURN.

For deletions, find the end of the string, and delete with either DELETE, RUBOUT or CONTROL H, depending on the type of terminal. New material may then be added if desired; if not hit RETURN.

The CONTROL Fe was entered to find the "e" in "The".

The CONTROL Fl was enterd twice to find the second "l" in "all".

The "*" was used to permit interaction in case the string being searched for had multiple occurences, and replacement was to be on a selective basis. The "23" is the count (in hex) to the start of the string /man/ in line 0010. The "a" is user approval to alter; entry of "s" would skip the alteration.

When editing is completed, enter MANUSCRIPT SET, to inhibit line number printing, and print the final copy. The process is less complicated than it would appear from the example, and will soon become almost automatic; the user will see, almost at once, simpler, though less illustrative, means for accomplishing the editing above.

It is good operating procedure to have a backup copy of the material which is being edited on tape, in case of operator errors with the MO, CO, DE, etc. commands.

3.4 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.

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 terminal input buffer is 80 characters in length. There are 9 tab points preset at 8 character intervals. Thus, the first tab point is at the 8th column, the second at the 16th column, etc. Entry of TAB or CONTROL 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 exit from this mode, type ≥//.

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 8th column is formed. 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 different from the same label entered as lower case.

SECTION 4.0

ASSEMBLER (ASM)

4.1 ASSEMBLER FEATURES

The ASM scans the source program in the text file. This requires at least two passes (or scans). On the first pass, the ASM generates a label file (or symbol table) and outputs any errors that may occur. On the second pass the ASM 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 unit 0 and may be reloaded into the memory using the relocating loader at practically any location.

4.2 SOURCE STATEMENT SYNTAX

Each source statement consists of five 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 set plus the following characters:

. / 0 thru 9 : ; < > ?

The label is entered immediately after the line number or prompt (≥) 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 from Table A or pseudo op from 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 or tabs 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 may be entered if a semicolon (;) follows the line number or ≥ if in auto line numbering mode.

For converting 6502 assembly language programs written on the System 65 or on MOS Technology Timesharing Cross Assembler, refer to Appendex C.

TABLE A - 6502 MNEMONICS

(For a description of each mnemonic, consult the MNA-2 SY6500 Programming Manual)

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

TABLE B - PSEUDO OPS

NAME	EXAMPLE	PURPOSE/USE
.BA expression	.BA $200	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.
.BY	.BY 00 'ABCD' 47 69 'Z' $FC %1101	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 (').
.CE	.CE	Continue assembly if errors other than !07, !04, or !17 occur. All error messages will be printed.
.CT	.CT	Indicates that the source program continues to tape.
label .DE expression	.DE INDEV IN .DE INDEV	Assign the address calculated from the expression to the label. Designate as external and put in label file. An error will result if the label is omitted.
label .DI expression	.DI TABLE ASCII .DI TABLE	Assign the address calculated from the expression to the label. Designate as internal and put in label file. An error will result if the label is omitted.
.DS expression	.DS 20 .DS $00F0	Define a block of storage. For example, if expression equated to 4, then ASM will skip over 4 bytes. **Note:** The initial contents of the block of storage are undefined.
.EC	.EC	Suppress output of macro generated object code on source listing. See Section 4.7. This is the default condition.
.EJ	.EJ	Eject to top of next page if ≥HA SET was previously entered.

TABLE B - PSEUDO OPS (CONT)

NAME	EXAMPLE	PURPOSE/USE
.EN	.EN	Indicates the end of the source program.
.ES	.ES	Output macro generated object code on source listing. See Section 4.7.
.LC	.LC	Clear the list option so that the assembly terminates printing the source listing after the .LC on pass 2.
.LS	.LS	Set the list option so that the assembly begins printing out the source listing after the .LS on pass 2.
!!!label .MD (p1 p2 p3...)		Macro definition. See Section 4.7.
.MC expression	.MC $700 .MC CAT .MC ORIGIN+$1000	When storing object code, move code to the address calculated from the expression but assemble in relation to that specified by the .BA pseudo op. An undefined address results in an immediate assembly halt.
.ME	.ME	Macro end. See Section 4.7.
.OC	.OC	Clear the object store option so that object code after the .OC is not stored in memory. This is the default option.
.OS	.OS	Set the object store option so that object code after the .OS is stored in memory on pass 2.
.RC	.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.

TABLE B - PSEUDO OPS (CONT)

NAME	EXAMPLE	PURPOSE/USE
.RS	.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.
.SE expression	.SE BASIC .SE $C000	Store the address calculated from the 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.
.SI expression	.SI START .SI TABLE .SI =+4	Store the address calculated from the expression in the next two memory locations. Consider this address as being an internal address.

NOTE

Labels may be entered with any of the pseudo ops, but are mandatory where indicated.

TABLE C - EXPRESSIONS

An expression must not contain embedded spaces and is constructed from the following:

Symbolic Labels:

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

Constants:

Decimal, hex, or binary values may be entered. If no special symbol precedes the numerials then the RAE assumes decimal (example: 147). If $ precedes then hex is assumed (example: $F3). Only the last four hex digits are used. If % precedes then binary is assumed (example: %11001). Leading zeros do not have to be entered. All numbers greater than 65,536 are reduced modulo 2^{16}.

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 expressions follow:

LDA	#%1101	load immediate 00001101
STA	*TEMP+$01	store at byte following TEMP (Zero page)
LDA	$471E36	load from $1E36 (47 is ignored)
JMP	LOOP+C-$461	
BNE	=+8	branch to current PC plus 8 bytes (current PC is first byte of next instruction)

One reserved symbol is A, as in ASL A. The letter A followed with a space in the operand field indicates accumulator addressing mode.

ASL A+$00 does not result in accumulator addressing but instead references a memory location.

TABLE D - ADDRESSING MODE FORMAT

Immediate

LDA #%1101	binary 00001101, the pound sign (#) indicates immediate addressing
LDA #$F3	hex F3
LDA #F3	load value of label F3
LDA #'A	ASCII A
LDA #H,expression	hi part of the value of the expression
LDA #L,expression	lo part of the value of the expression

Absolute

LDA expression

Zero Page

LDA *expression	the asterisk (*) indicates zero page addressing

Absolute Indexed

LDA expression,X

LDA expression,Y

Zero Page Indexed

LDA *expression,X

LDX *expression,Y

Indexed Indirect

LDA (expression,X)

Indirect Indexed

LDA (expression),Y

Indirect

JMP (expression)

TABLE D - ADDRESSING MODE FORMAT (CONT)

Accumulator

ASL A	letter A indicates accumulator addressing mode

Implied

TAX CLC	operand field ignored

Relative

```
BEQ expression
```

4.3 LABEL FILE (OR SYMBOL TABLE)

A label file is constructed by the assembler and may be outputted at the end of assembly (if an .LC pseudo op was not encountered) or via the ≥LA command. The output consists 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 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 .BA, and zzzz is last address in relation to .MC.

4.4 ASSEMBLING FROM MEMORY

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 is encountered, the ≥PA command would have to be executed to perform the second pass.

4.5 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 numbers assigned to the modules are ignored. NOTE: SYM users should refer to Section 10.0, paragraph 4, before assembling from tape.

Source statements within a module will be assembled in numerical order but the modules will be assembled in the order in which they are encountered. Source statement numbering is restarted for each module. If a line number is specified in the >AS command indicating the start of assembly, it applies for all modules.

The ASM 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 (!07 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 unit and type >AS x. Either way the ASM will start and stop tape unit 1 until the .EN pseudo op is encountered. At that point tape unit 1 is turned off, and the message RDY. FOR PASS 2 is outputted.

RAE is now in the TED mode. Rewind the tape unit (>ON 1 and >OFF 1 accordingly). Perform 1 or 2 as described above and type >PASS to perform the second pass. Again tape unit 1 will be turned on and off accordingly under control of the ASM software.

4.6 CREATING A RELOCATABLE OBJECT FILE

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. Constants (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 .SE stores a two byte external address and .SI stores a two byte internal address. Similarily the relocating loader will alter the internal address and not the external address.

An example of an external address would be the calls to the system 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 unit 0 and ready. If the entire source program is in memory, simply type ≥OU.

If the source program is on tape type ≥OU, the ASM will turn both tape units on and off until the end of assembly. The relocatable object file will be recorded on the tape in unit 0.

After the relocatble object file has been recorded, record an end of file mark via the ≥PU X command.

4.7 MACROS

RAE provides 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 INCD (VALUE.1). 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 ASM to process it. A macro is defined via the .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:

```
!!!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 that 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  (COUN+2)
```

or any other labels or expressions you may choose.

NOTE

In the expansion of INCD the code to increment the variable LOC is not being generated; instead the code to increment the associated variable in the call for expansion. Also parentheses must be used with the parameter labels both in the definition and in the call.

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:

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

The only difference is ...SKIP is substituted for SKIP. What the ASM 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 ASM 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 are each 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. 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 definition 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 non-symbolic 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. If .EC was entered, only the line which contained the macro call will be output in the source listing.

4.8 CONDITIONAL ASSEMBLY

ASM 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 ASM conditional assembly feature to assemble code concerned with one of the character densities.

Before we continue with this example, let us describe the conditional assembly operators:

IFE expression	If the expression equates to a zero quantity, then assemble to end of control block.
IFN expression	If the expression equates to a non-zero quantity then assemble to end of control block.
IFP expression	If the expression equates to a positive quantity (or 0000), then assemble to end of control block.
IFM expression	If the expression equates to a negative (minus) quantity, then assemble to end of control block.
***	Three asterisks in the mnemonic field indicates the end of the control block.
SET symbol = expression	Set the previously defined symbol to the quantity calculated from the expression.

NOTE

All expressions are evaluated 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, with SET CHAR.LINE = 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         ...SKIP
;SORT CODE FOLLOWS
SORT.CALL
           .
           .
           .
           .
           .
           .
           RTS
...SKIP    SET EXPAND = 1
           ***
           .ME
```

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. 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.

4.9 ASSEMBLER DEFAULT PARAMETERS

- Assumes assembling from memory (otherwise use .CT)
- Does not store object code in memory (otherwise use .OS)
- Begins assembly at $0200 (otherwise use .BA)
- Output listing clear (otherwise use .LS OR ≥ASSEMBLE LIST)
- Stops assembly on errors (otherwise use .CE)
- Stores object code beginning at $0200 unless a .BA or .MC is encountered and if .OS is present.
- Generates relocatable address
- Macro object code is not output (otherwise use .ES)

SECTION 5.0

RELOCATING LOADER

A source listing of the relocating loader (Appendix D) is provided. The relocating loader is not part of the RAE program body, and the user will have to enter it via the listing.

If you prefer to have the loader 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 an .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 $00E0 (lo) and $00E1 (hi), the object file number into $0110, the relocatable buffer address in 00C8 (lo) and 00C9 (hi) and then start execution at $0200.

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 the system monitor. The contents of register A indicates the following:

> 00 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 $00E0 and $00E1 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:

> 00C8-00C9, 00DC-00E1
> 0110, 011E-0121, 017A-0184

plus other stack area for subroutine control.

SECTION 6.0

FILE NUMBERS

Information to be recorded on or read from tape via the ≥PU, ≥GE, and ≥OU commands may be assigned a file identification number to distinguish between files. 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 ≥GE, 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
F00 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.

SECTION 7.0

ERROR CODES

An error message of the form !xx AT LINE yyyy/zz where xx is the error code, yyyy 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:

17	Checksum error on tape load
16	Illegal tape unit number
15	Syntax error in ≥ED command
14	Cannot generate relocatable object tape with errors or no previous assembly
11	Missing parameter in ≥NU command
10	Overflow in line # renumbering **CAUTION: You must properly renumber the text file or part of the file may be deleted by subsequent operations.**
0F	Overflow in text file - line not inserted
0E	Overflow in label file - label not inserted
0D	Expected hex characters, found none
0C	Illegal character in label
0B	Unimplemented addressing mode
0A	Error in or no operand
09	Found illegal character in decimal string
08	Undefined label (may be illegal label)
07	.EN pseudo op missing
06	Duplicate label
05	Label missing in .DE or .DI pseudo op
04	.BA or .MC operand undefined
03	Illegal pseudo op
02	Illegal mnemonic
01	Branch out of range
00	Not a zero page address
ED	Error in command input

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

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	Number of macro reference parameters is different from the number of macro dummy parameters 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	Attempted expansion before definition

SECTION 9.0

SPECIAL NOTES

- In addition to the program memory space the RAE uses the following memory locations:

  ```
  0100 - up    depending on type of function
  00B6 - 00FF    reserved for RAE and system monitor
  ```

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

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

- When entering source modules (without .EN) you can perform a short test on the module by assembling the module while in the text file and looking for the !07 error. If other error messages occur, you have errors in the module. This short test 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 outputting an assembly listing in order to provide a neat printout.

- 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. These locations contain invalid data until after the first assembly. 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).

- 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 00DD, 00DE. This is an address which points to the end of the outputted line.

SECTION 10.0

SPECIFIC APPLICATION NOTES

1. The default file boundaries for RAE are: test file = 0200-0BFC, label file = 0C00-0EFC, and relocatable buffer = 0F00. When entering the file boundary via the SET command, enter the end address minus 3.

   Example: If the end = 0BFF, then enter 0BFC.

2. RAE provides software for controlling two tape motors. RAE assumes the record unit (unit 0) is connected to the SYM motor control. If the user implements motor control hardware for the play unit (unit 1), RAE can control it via APB7, pin A-15 ("1" = off, "0" = on).

3. MDT1000 has both unit 0 and unit 1 remote motor control hardware as standard hardware.

4. The following must exist for installation of RAE-1/2 (two 4K ROMs) into SYM.

   RAE P/N 02-0023 inserted into socket U22 and
   RAE P/N 02-0024 inserted into socket U23

   The jumpers must be configured as follows:

C-1	H-3
D-1	L-46, 46*
G-2	M-15, 16

5. The following must exist for installation of RAE-1 (one 8K ROM) into SYM.

   RAE-1 (P/N 02-0053A) in socket U23

   The jumpers must be configured as follows:

   D-1
   M-15, 16, 46, 47*

   Add the following:

   Jumper 4 to U2-1
   Jumper H to U2-2

   (U2 is an inverter located to the right of logo)

   RAE-1 (P/N 02-0053B) in socket U23

   The jumpers must be configured as follows:

   D-1
   H-4
   M-15, 16, 46, 47*

   For both versions, remove jumper from D to 3 and also jumper from H to 6.

6. A manually-entered patch is required for RAE-1 V1.0 when used on SYM for assembly from cassette tape. The user must enter a flag and a vector into zero page. The patch may be stored any place in RAM which does not conflict with RAE-1, SYM-1, or application software. Since RAE-1 cold start entry clears the flag to zero, the patch must be entered after first transferring control to RAE-1 and then exiting RAE-1.

* In early versions of SYM-1, jumper points 46 and 47 are not labelled. For these boards, jumper points 46 and 47 are identical to U10-7 and U10-9, respectively.

The patch shown below is placed at the end of the default label file.

LOCATION	CONTENT	COMMENT
EE	01	Enter flag
F6	F5	Enter vector to
F7	0E	. . .patch
EF5	AD	Patch is 3
EF6	11	. . .instructions
EF7	01	
EF8	D0	
EF9	03	
EFA	8D	Store 0 into
EFB	10	location $110
EFC	01	
EFD	4C	Jump back into
EFE	68	RAE-1
EFF	EF	

To install the patch, perform the following:

1.	Enter RAE-1	Type: G B000↵
2.	Exit RAE	Type: BR↵
3.	Use M command three times to modify EE, F6-F7, and EF8-EFF	
4.	Return to RAE <u>warm</u> entry	Type: G↵

CORRECT

The patch shown below is placed at the end of the default label file.

LOCATION	CONTENT	COMMENT
EE	01	Enter flag
F6	A0	Enter vector to
F7	00	. . .patch
A0	A9	Patch is 3
A1	00	. . .instructions
A2	8D	Store 0 into
A3	10	location $110
A4	01	
A5	4C	Jump back into
A6	68 [illegible]	RAE-1
A7	EF	

DISC IN VECTOR

To install the patch, perform the following:

1. Enter RAE-1 Type: G B000↲
2. Exit RAE Type: BR↲
3. Use M command three times to modify EE, F6-F7, and ~~EF8-EFF~~ A0 - A7
4. Return to RAE <u>warm</u> entry Type: G↲

EE
F6,F7

```
A9 00
8D 10 01
4C 68 EF     JMP
```

RAE Initializes: B7 to 60
CE to 00 0F
D3 to 00 02 00 0C
DB to 01
E3 to 00 00 00
? EA to 08
EE to 00 00

Purple seems only to make current file number 0

Appendices

APPENDIX A

ASCII CHARACTER CODES

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
000	000	NUL- ↑@	043	02B	+	086	056	V
001	001	SOH- ↑A	044	02C	,	087	057	W
002	002	STX- ↑B	045	02D	-	088	058	X
003	003	ETX- ↑C	046	02E	.	089	059	Y
004	004	EOT- ↑D	047	02F	/	090	05A	Z
005	005	ENG- ↑E	048	030	0	091	05B	[
006	006	ACK- ↑F	049	031	1	092	05C	\
007	007	BEL- ↑G	050	032	2	093	05D	]
008	008	BS- ↑H	051	033	3	094	05E	↑
009	009	HT- ↑I	052	034	4	095	05F	←
010	00A	LF- ↑J	053	035	5	096	060	
011	00B	VT- ↑K	054	036	6	097	061	a
012	00C	FF- ↑L	055	037	7	098	062	b
013	00D	CR- ↑M	056	038	8	099	063	c
014	00E	SO- ↑N	057	039	9	100	064	d
015	00F	SI- ↑O	058	03A	:	101	065	e
016	010	DLE- ↑P	059	03B	;	102	066	f
017	011	DC1- ↑Q	060	03C	<	113	067	g
018	012	DC2- ↑R	061	03D	=	104	068	h
019	013	DC3- ↑S	062	03E	>	105	069	i
020	014	DC4- ↑T	063	03F	?	106	06A	j
021	015	NAK- ↑U	064	040	@	107	06B	k
022	016	SYN- ↑V	065	041	A	108	06C	l
023	017	ETB- ↑W	066	042	B	109	06D	m
024	018	CAN- ↑X	067	043	C	110	06E	n
025	019	EM- ↑Y	068	044	D	111	06F	o
026	01A	SUB- ↑Z	069	045	E	112	070	p
027	01B	ESC- ↑[	070	046	F	113	071	q
028	01C	FS- ↑\	071	047	G	114	072	r
029	01D	GS- ↑]	072	048	H	115	073	s
030	01E	RS- ↑↑	073	049	I	116	074	t
031	01F	VS- ↑←	074	04A	J	117	075	u
032	020	SPC-	075	04B	K	118	076	v
033	021	!	076	04C	L	119	077	w
034	022	"	077	04D	M	120	078	x
035	023	#	078	04E	N	121	079	y
036	024	$	079	04F	O	122	07A	z
037	025	%	080	050	P	123	07B	
038	026	&	081	051	Q	124	07C	'
039	027	'	082	052	R	125	07D	
040	028	(	083	053	S	126	07E	~
041	029	)	084	054	T	127	07F	DEL
042	02A	*	085	055	U			

LF = Line Feed FF = Form Feed CR = Carriage Return

DEL = Rubout ↑ = Control Key

APPENDIX B

RAE I/O LINKAGES

The following describes user I/O linkages and page 0 (zero) vectors. Functions described include CRT, keyboard, break key, printer, CONTROL Y, and user. Page 0 (zero) locations $EC - $F7 are reserved for future RAE extensions.

BREAK KEY	RAE vectors thru INSVEC ($A666) in system RAM for testing for the break key being depressed. This 3-byte location contains a JMP instruction. If you wish to substitute another routine to detect if the break key is depressed, change the 2-byte address part of the JMP instruction to point to the alternate break key processing routine.
CONTROL Y	When a CONTROL Y (↑Y) is entered, RAE "JUMPS" to location $0000 for execution of user supplied instructions. RAE does not enter any default code at this location. The user supplied routine can reenter RAE via a JMP to the warm start address ($B003). None of the registers need be preserved.
CRT	RAE vectors thru OUTVEC ($A663) in system RAM for outputting to the CRT. This 3-byte location contains a JMP instruction. If you wish to redirect output to another device such as a printer, change the 2-byte address part of the JMP instruction to point to the alternate devices software driver.
KEYBOARD	RAE vectors thru INVEC ($A660) in system RAM for inputting from the keyboard. This 3-byte location contains a JMP instruction. If you wish to redirect output to another device such as a TTY, change the 2-byte address part of the JMP instruction to point to the alternate devices software driver.
PRINTER	RAE reserves 3-bytes starting at $00B6 which the user can use to vector to a routine which drives an alternate output device. On cold start, RAE enters an RTS instruction at this vector. When an ≥HA SET command is initiated, program control is transferred thru this vector for driving an alternate output device while outputting to the CRT at the same time. Register A will contain the ASCII character to be output. Registers X and Y should be preserved and the decimal mode bit in the PSR should be left cleared. Outputting thru this vector is terminated via ≥HA CLEAR.
USER	When a user command is entered, RAE "JUMPS" to location $0003 for execution of user supplied instructions. RAE does not enter any default code at this location. The user supplied routine can reenter RAE via a JMP to the warm start address ($B003). None of the registers need be preserved.

APPENDIX C

CONVERTING MOS TECHNOLOGY/SYSTEM 65 ASSEMBLY LANGUAGE PROGRAMS TO RAE

This table shows by example, how to translate from MOS Technology/System 65 syntax to RAE syntax.

LINE NO.	MOS TECHNOLOGY/ SYSTEM 65		RAE	
.0006		* = $A600		.BA $A600
.0007	SCPBUF	* = *+$20	SCPBUF	.DS $20
.0008	RAM	= *	RAM	.DI = ;or just RAM
.0024	RC	= SCRD	RC	.DI SCRD
.0087	PADA	= $A400	PADA	.DE $A400
.0252		.BYT $FF,$FF,$FF		.BY $FF $FF $FF
.0430		BNE *+5		.BNE = +5
.0434		STX $FF		STX *$FF
.1589	ASCM1	= *-1	ASCM1	.DI = -1
.1711	STDVAL	.DBY $D54C,$2410	STDVAL	.BY $D5 $4C
.1723		.WORD $C000		.SE $C000
.1724		.WORD TTY		.SI TTY
.1778		.END		.EN
		.LDA #>EXPRESSION		LDA #H,EXPRESSION
		.LDA #<EXPRESSION		LDA #L,EXPRESSION

1. The following RAE directives do not have equivalent functions in the MOS Technology/System 65 assembler:

   .LS, .LC, .OS, .OC, .RC, .MD, .ME, .EC, .ES, .RS

2. The following RAE directives have similar functions in the System 65 assembler:

   .CE is an intrinsic attribute of the System 65 assembler.

   .EJ is ".PAG" in the System 65 assembler.

   .MC is implemented in a constrained fashion on System 65 i.e., the "entire" object program may be assembled into a different memory space than the one specified for execution.

   .CT is available on System 65 for source stored in multiple disk files via the FILE directive.

APPEDIX D

```
0010 ;****RELOCATING LOADER FOR SYNERTEK SYSTEMS RAE-1
0020 ;
0030 ;
0040 ;
0050            .OS
0060 ;
0070 ;***COPYRIGHT 1979 BY SYNERTEK SYSTEMS CORP.***
0080 ;***    ALL RIGHTS RESERVED.                ***
0090 ;
0100 ;
0110 ;
0120 ;
0130 ;++++++  USER INPUTTED VARIABLES BEFORE EXECUTION  ++++++
0140 FILE/NO    .DE $0110    ;FILE NUMBER (0-99)
0150 OFFSET     .DE $E0      ;RELOCATOR OFFSET (2 BYTES)
0160 BUFFER     .DE $C8      ;ADDRS. OF R.L. BUFFER
0170 ;
0180 ;
0190 ;
0200 ;          RELOCATOR DIRECTIVES
0210 ;
0220 ;  DIRECTIVE               DESCRIPTION
0230 ;  ---------               -----------
0240 ;    0F              EXTERNAL 2 BYTE ADDRS. PRECEEDS,
0250 ;                   DON'T RELOCATE. OTHERWISE RELOCATE.
0260 ;
0270 ;    1F              #L,  DATA PRECEEDS.
0280 ;
0290 ;    2F              #H,  DATA PRECEEDS, LO PART FOLLOWS.
0300 ;
0310 ;    3F              .AS OR .HS BYTE FOLLOWS.
0320 ;
0330 ;    4F              .SE OR .SI 2 BYTE ADDRS. FOLLOWS.
0340 ;
0350 ;    5F              TURN RELOCATOR ON (VIA .RS).
0360 ;                   (RESOLVE ADDRESSES AND RELOCATE CODE)
0370 ;
0380 ;    6F              TURN RELOCATOR OFF (VIA .RC).
0390 ;                   (RESOLVE ADDRESSES BUT DO NOT
0400 ;                    RELOCATE CODE)
0410 ;
0420 ;    7F              ,DS - 2 BYTE BLOCK VALUE FMLLMUS,
0430 ;
0440 ;
0450            .BA $0200
0460 ;
0470 ;TAPE INPUT PARMS
0480 LOAD/NO    .DE $0180 0: NO STORE; 1: STORE
0490 TSTART     .DE $A64C LOAD BEGINNING AT TSTART
0500 TEND       .DE $A64A STOP LOADING AT TEND
0510 ;
0520 ;
0530 ;HEADER INPUT DATA
0540 HFILE/NO   .DE $017A HEADER FILE NUMBER
0550 HSTART     .DE $017B HEADER START
0560 HEND       .DE $017D HEADER END
0570 ;
0580 ;
```

```
               ;VARIABLES
               SCRAT       .DE $11E SCRATCH AREA
               TEMP1       .DE $11F SCRATCH AREA
               TEMP2       .DE $120 SCRATCH AREA
               SAVE        .DE $121 SCRATCH AREA
               ADDRS       .DE $DC 4 BYTES OF ADDRESS INFO.
               BUFF.END    .DE $0123 END OF 256 BYTE BUFFER
               BUFF.INDEX  .DE $0124 PRESENT ACCESSED DATA FROM BUFFER
               ;
               ;
               ;R(X)=00:   RELOCATOR ON
               ;R(X)=02:   RELOCATOR OFF
               ;
               ;BEGIN EXECUTION AT LABEL START
               ;
0200- A2 FF    START       LDX #$FF
0202- 9A                   TXS INITIALIZE STACK
0203- E8                   INX R(X)=00: SET RELOCATOR INITIALLY TO ON
0204- 20 86 8B             JSR ACCESS
0207- D8                   CLD
0208- 8E 21 01             STX SAVE R(X)=00
020B- 20 E6 02             JSR LOAD<BUFF
020E- 4C 14 02             JMP ENTY
0211- 20 74 03 LOOP1       JSR GET<DATA
               ;
0214- C9 7F    ENTY        CMP #$7F      ;CKG. FOR .DS
0216- D0 03                BNE PRO.SF
0218- 4C AA 03             JMP PRO.7F    ;JUMTO PROCESS DIR. 7F
021B- C9 3F    PRO.SF      CMP #$3F CKG. FOR RELOCATOR DIRECTIVE
021D- D0 0B                BNE OP<CKG
021F- 20 74 03             JSR GET<DATA
0222- 81 DC                STA (ADDRS,X)
0224- 20 88 03             JSR INC<ADDRS
0227- 4C 11 02             JMP LOOP1
022A- C9 4F    OP<CKG      CMP #$4F CKG. FOR .SE, .SI
022C- D0 03                BNE W:
022E- 4C AD 02             JMP TWO<BYT<AD
0231- C9 5F    W:          CMP #$5F CKG. FOR RELOCATOR ON
0233- D0 04                BNE CKNX
0235- A2 00                LDX #$00
0237- F0 D8                BEQ LOOP1
               ;
0239- C9 6F    CKNX        CMP #$6F CKG. FOR RELOCATOR OFF
023B- D0 04                BNE NO<REL
023D- A2 02                LDX #$02
023F- D0 D0                BNE LOOP1
0241- 81 DC    NO<REL      STA (ADDRS,X) STORE OP CODE
0243- 20 88 03             JSR INC<ADDRS
0246- C9 00                CMP #$00 CKG.
0248- F0 C7                BEQ LOOP1
024A- C9 20                CMP #$20 CKG. FOR JSR INSTR.
024C- F0 5F                BEQ TWO<BYT<AD
024E- 8D 21 01             STA SAVE SAVE R(A), IT CONTAINS OP CODE
0251- 29 9F                AND #$9F
0253- F0 BC                BEQ LOOP1
0255- AD 21 01             LDA SAVE RESTORE OP CODE
0258- 29 1D                AND #$1D
025A- C9 08                CMP #$08 CKG. FOR ONE BYTE INSTR.
```

```
025C- F0 B3                   BEQ LOOP1
025E- C9 18                   CMP #$18 CKG. FOR ONE BYTE INSTR.
0260- F0 AF                   BEQ LOOP1
              ;
              ;NOW, TEST FOR INSTR. CONTAINING 2 BYTES
              ;OF ADDRESS INFORMATION
              ;
0262- AD 21 01                LDA SAVE RESTORE OP CODE
0265- 29 1C                   AND #$1C
0267- C9 1C                   CMP #$1C
0269- F0 42                   BEQ TWO<BYT<AD
026B- C9 18                   CMP #$18
026D- F0 3E                   BEQ TWO<BYT<AD
026F- C9 0C                   CMP #$0C
0271- F0 3A                   BEQ TWO<BYT<AD
              ;
              ;THE REMAINING CONTAIN ONE BYTE OF
              ;ADDRESS INFORMATION
              ;
              ;PROCESSING OF ONE BYTE ADDRESSES AND IMMEDIATE DATA
              ;
0273- 20 74 03 ONE<BYT<AD     JSR GET<DATA
0276- 81 DC                   STA (ADDRS,X)
0278- 20 88 03                JSR INC<ADDRS
027B- 20 74 03                JSR GET<DATA
027E- C9 2F                   CMP #$2F CKG. FOR RELOCATOR DIRECTIVE
0280- F0 14                   BEQ IMM<HI CKG. FOR #H,
0282- C9 1F                   CMP #$1F CKG. FOR RELOCATOR DIRECTIVE
0284- D0 8E                   BNE ENTY
              ;
              ;PROCESS #L, DATA FOR RELOCATION
0286- 20 95 03 IMM<LO         JSR DEC<ADDRS
0289- 18                      CLC
028A- A1 DC                   LDA (ADDRS,X)
028C- 65 E0                   ADC *OFFSET+00 ADD OFFSET LOW PART FOR #L,
028E- 81 DC                   STA (ADDRS,X)
0290- 20 88 03                JSR INC<ADDRS
0293- 4C 11 02 BACK<TO<L1     JMP LOOP1
              ;PROCESS #H, DATA FOR RELOCATION
0296- 20 74 03 IMM<HI         JSR GET<DATA LOW BYTE FOLLOWS REL. DIR.
0299- 18                      CLC
029A- 65 E0                   ADC *OFFSET FROM THE LO ADDRS. PART
029C- 08                      PHP
029D- 20 95 03                JSR DEC<ADDRS
02A0- 28                      PLP
02A1- A1 DC                   LDA (ADDRS,X)
02A3- 65 E1                   ADC *OFFSET+$1 NOW FORM THE EFFECTIVE #H,
02A5- 81 DC                   STA (ADDRS,X)
02A7- 20 88 03                JSR INC<ADDRS
02AA- 4C 11 02                JMP LOOP1
              ;
              ;PROCESSING OF TWO BYTE ADDRESSES
02AD- A0 02    TWO<BYT<AD     LDY #$02
02AF- 98       XX             TYA
02B0- 48                      PHA SAVE R(Y)
02B1- 20 74 03                JSR GET<DATA
02B4- 81 DC                   STA (ADDRS,X)
02B6- 20 88 03                JSR INC<ADDRS
```

```
02B9- 68                      PLA
02BA- A8                      TAY RESTORE R(Y)
02BB- 88                      DEY
02BC- D0 F1                   BNE XX
02BE- 20 74 03                JSR GET<DATA
02C1- C9 0F                   CMP #$0F CKG. FOR RELOCATOR DIRECTIVE
02C3- D0 03                   BNE XY
02C5- 4C 11 02                JMP LOOP1
02C8- 48          XY          PHA
02C9- 20 95 03                JSR DEC<ADDRS
02CC- 20 95 03                JSR DEC<ADDRS
                  ;DECREMENT  BACK TO ADDRESS START
                  ;
02CF- A1 DC                   LDA (ADDRS,X)
02D1- 18                      CLC
02D2- 65 E0                   ADC *OFFSET AND OFFSET LO
02D4- 81 DC                   STA (ADDRS,X)
02D6- 20 88 03                JSR INC<ADDRS
02D9- A1 DC                   LDA (ADDRS,X)
02DB- 65 E1                   ADC *OFFSET+$1 ADD OFFSET HI
02DD- 81 DC                   STA (ADDRS,X)
02DF- 20 88 03                JSR INC<ADDRS
02E2- 68                      PLA
02E3- 4C 14 02                JMP ENTY
                  ;
                  ;SUBROUTINE LOAD BUFFER WITH DATA FROM TAPE
                  ;
02E6- A9 7A       LOAD<BUFF   LDA #$7A ADDLO OF START OF HEADER
02E8- 8D 4C A6                STA TSTART+$00
02EB- A9 7F                   LDA #$7F ADDLO OF END OF HEADER
02ED- 8D 4A A6                STA TEND+$00
02F0- A9 01                   LDA #$01 HI ADDRS
02F2- 8D 4D A6                STA TSTART+$01
02F5- 8D 4B A6                STA TEND+$01
02F8- 8D 80 01                STA LOAD/NO 01: INDICATE TO LOAD
02FB- 20 D5 03                JSR USER/LOAD USER LDA↑BD FROM
                                                         TAPE ROUTINE
                  ;
                  ;THE ABOVE SETS UP AND LOADS HEADER INFORMATION
                  ;FROM TAPE.  THE HEADER CONTAINS THE MODULE FILE
                  ;NUMBER, AND STARTING AND ENDING ADDRESSES OF
                  ;FOLLOWING DATA.
                  ;
                  ;
02FE- D0 4D                   BNE ERROR IF Z-BIT FALSE, THEN
                                                         ERROR IN LOADING
0300- A2 00                   LDX #$00
                  ;
0302- AD 7D 01                LDA HEND+$00
0305- 38                      SEC
0306- ED 7B 01                SBC HSTART+$00
                  ;CALCULATE NUMBER OF BYTES IN FOLLOWING DATA
                  ;
0309- 8D 23 01                STA BUFF.END INITIALIZE BUFFER END
030C- AD 7E 01                LDA HEND+$01                  POINTER
030F- ED 7C 01                SBC HSTART+$01
0312- D0 39                   BNE ERROR ONLY 256 BYTE BUFFER ALLOWED
0314- A5 C8                   LDA *BUFFER
0316- 8D 4C A6                STA TSTART
0319- 18                      CLC
```

```
031A- 6D 23 01             ADC BUFF.END # BYTES
031D- 8D 4A A6             STA TEND
0320- A5 C9                LDA *BUFFER+01
0322- 8D 4D A6             STA TSTART+$01
0325- 69 00                ADC #$00
0327- 8D 4B A6             STA TEND+$01
                ;NOW THE START AND END ADDRESS PARMS HAVE BEEN
                ;SET UP TO LOAD FROM TAPE INTO THE BUFFER.
                ;
032A- AD 10 01             LDA FILE/NO USER ENTERED FILE NUMBER
032D- F0 08                BEQ STORE.DATA IF F# = 00. LOAD ANYWAY
032F- CD 7A 01             CMP HFILE/NO CMP WITH USER VERSUS THAT
                                                      ON TAPE
0332- F0 03                BEQ STORE.DATA
0334- 8E 80 01             STX LOAD/NO R(X)=0; NO STORE
0337- 20 D5 03  STORE.DATA JSR USER/LOAD
                ;
                ;THE ABOVE LOADS IN DATA INTO BUFFER DEPENDING
                ;ON THE STATE OF LOAD/NO
                ;
033A- D0 11                BNE ERROR Z-BIT = FALSE THEN ERROR
033C- A2 00                LDX #$00
033E- AD 7A 01             LDA HFILE/NO
0341- C9 EE                CMP #$EE COMPARE IF END OF FILE
0343- D0 0C                BNE BUFFLOADED
0345- A9 00                LDA #$00 INDICATE GOOD LOAD
0347- 00        B          BRK
0348- EA                   NOP
0349- EA                   NOP
034A- 4C 00 02             JMP START
034D- A9 EE     ERROR      LDA #$EE INDICATE ERROR IN LOAD
034F- D0 F6                BNE B
                ;
                ;
                ;NOW GET ADDRS. INFO AND PUT IN ADDRS+$2. +$3
                ;ADDRS. INFO. IS IN FIRST TWO BYTES OF BUFFER
                ;
0351- AD 80 01  BUFFLOADED LDA LOAD/NO CKG. IF PROPER DATA
0354- F0 90                BEQ LOAD<BUFF
0356- AE 21 01             LDX SAVE RESTORE R(X)
0359- A0 00                LDY #$0
035B- B1 C8                LDA (BUFFER),Y
035D- 85 DE                STA *ADDRS+$2
035F- C8                   INY
0360- B1 C8                LDA (BUFFER),Y
0362- 85 DF                STA *ADDRS+$3
0364- 8C 24 01             STY BUFF.INDEX SET BUFFER DATA POINTER
                ;
                ;SET RELOCATION ADDRS. IN ADDRS+$0, +$1
0367- A5 DE                LDA *ADDRS+$2
0369- 18                   CLC
036A- 65 E0                ADC *OFFSET
036C- 85 DC                STA *ADDRS
036E- A5 E1                LDA *OFFSET+$1
0370- 65 DF                ADC *ADDRS+$3
0372- 85 DD                STA *ADDRS+$1
                ;
0374- 8E 21 01  GET<DATA   STX SAVE X IN CASE WE BR. TO LOAD<BUFF
0377- EE 24 01             INC BUFF.INDEX INC. 256 BYTE BUFFER
                                                      POINTER
```

```
037A- AC 24 01  2910              LDY BUFF.INDEX
037D- CC 23 01  2920              CPY BUFF.END
0380- 90 03     2930              BCC WX BR. IF NOT AT END OF
                                          DATA IN BUFFER
0382- 4C E6 02  2940              JMP LOAD<BUFF RELOAD BUFFER
0385- B1 C8     2950 WX           LDA (BUFFER),Y
0387- 60        2960              RTS
                2970 ;
                2980 ;
                2990 ;INCREMENT ADDRS+$0, +$1 AND ADDRS+$2, +$3
                3000 ;
0388- E6 DC     3010 INC<ADDRS   INC *ADDRS
038A- D0 02     3020              BNE SKIP<INC1
038C- E6 DD     3030              INC *ADDRS+$1
038E- E6 DE     3040 SKIP<INC1   INC *ADDRS+$2
0390- D0 02     3050              BNE SKIP<INC2
0392- E6 DF     3060              INC *ADDRS+$3
0394- 60        3070 SKIP<INC2   RTS
                3080 ;
                3090 ;
                3100 ;DECREMENT ADDRS+$0, +1 AND ADDRS+$2, +$3
                3110 ;
0395- C6 DC     3120 DEC<ADDRS   DEC *ADDRS
0397- A5 DC     3130              LDA *ADDRS
0399- C9 FF     3140              CMP #$FF
039B- D0 02     3150              BNE SKIP<DEC1
039D- C6 DD     3160              DEC *ADDRS+$1
039F- C6 DE     3170 SKIP<DEC1   DEC *ADDRS+$2
03A1- A5 DE     3180              LDA *ADDRS+$2
03A3- C9 FF     3190              CMP #$FF
03A5- D0 02     3200              BNE SKIP<DEC2
03A7- C6 DF     3210              DEC *ADDRS+$3
03A9- 60        3220 SKIP<DEC2   RTS
                3230 ;
                3240 ;
                3250 ;7F LO HI -- PCL PCH 7F LO HI
                3260 ;
03AA- 20 74 03  3270 PRO.7F      JSR GET<DATA
03AD- 48        3280              PHA  ;SAVE LO
03AE- 20 74 03  3290              JSR GET<DATA
03B1- A8        3300              TAY  ;SAVE HI IN R(Y)
03B2- AD 24 01  3310              LDA BUFF.INDEX
03B5- C9 05     3320              CMP #$05      ;NO PROC. IF <= 4
03B7- 90 18     3330              BCC NO.PROC
03B9- 18        3340 PROC.DS     CLC
03BA- 68        3350              PLA  ;GET LO
03BB- 48        3360              PHA
03BC- 65 DC     3370              ADC *ADDRS
03BE- 85 DC     3380              STA *ADDRS
03C0- 98        3390              TYA  ;GET HI
03C1- 65 DD     3400              ADC *ADDRS+1
03C3- 85 DD     3410              STA *ADDRS+1
03C5- 68        3420              PLA
03C6- 48        3430              PHA  ;GET LO
03C7- 18        3440              CLC
03C8- 65 DE     3450              ADC *ADDRS+2
03CA- 85 DE     3460              STA *ADDRS+2
03CC- 98        3470              TYA  ;GET HI
03CD- 65 DF     3480              ADC *ADDRS+3
```

```
03CF- 85 DF     3490              STA *ADDRS+3
03D1- 68        3500 NO.PROC      PLA
03D2- 4C 11 02  3510              JMP LOOP1
                3520 ;
                3530 ;
                3540 ;
                3550 ;      ***SYSTEM MONITOR CASSETTE INTERFACE***
                3560 ;
                3570 ;
                3580 ;
                3590 ;     DEFINITIONS:
                3600 SAVER        .DE $8188
                3610 ACCESS       .DE $8B86
                3620 ID           .DE $A64E
                3630 MODE         .DE $FD
                3640 CONFIG       .DE $89A5
                3650 ZERCK        .DE $832E
                3660 P2SCR        .DE $829C
                3670 LOADT        .DE $8C78
                3680 NACCESS      .DE $8B9C
                3690 RESXAF       .DE $81B8
                3700 ;
                3710 ;
03D5- 20 88 81  3720 USER/LOAD    JSR SAVER     ;SAVE REGISTERS
03D8- A9 FF     3730              LDA #$FF      ;ID=FF FOR USER RANGE
03DA- 8D 4E A6  3740              STA ID
03DD- A0 80     3750              LDY #$80
03DF- 84 FD     3760              STY *MODE     ;BIT 7=1 FOR H.S.
03E1- A9 09     3770              LDA #$09
03E3- 20 A5 89  3780              JSR CONFIG
03E6- 20 2E 83  3790              JSR ZERCK
03E9- 20 9C 82  3800              JSR P2SCR
03EC- 20 7B 8C  3810              JSR LOADT+$3    ;ENTRY IN TAPE LOAD
03EF- D8        3820              CLD
03F0- A9 00     3830              LDA #$00      ;Z-BIT = T
03F2- 90 02     3840              BCC SKPERRU/L
03F4- A9 01     3850              LDA #$01      ;Z-BIT = F
                3860 SKPERRU/L
03F6- 4C B8 81  3870              JMP RESXAF ;RESTORE REGS. EXCEPT A,PSR
                3880 ;
                3890 ;
                3900 END.PGM      .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
/SCRAT=011E                /TEMP1=011F                /TEMP2=0120
/SAVE=0121                 /ADDRS=00DC                /BUFF.END=0123
/BUFF.INDEX=0124           START=0200                 LOOP1=0211
ENTY=0214                  PRO.SF=021B                OP<CKG=022A
W:=0231                    CKNX=0239                  NO<REL=0241
ONE<BYT<AD=0273            IMM<LO=0286                BACK<TO<L1=0293
IMM<HI=0296                TWO<BYT<AD=02AD            XX=02AF
XY=02C8                    LOAD<BUFF=02E6             STORE.DATA=0337
B=0347                     ERROR=034D                 BUFFLOADED=0351
GET<DATA=0374              WX=0385                    INC<ADDRS=0388
SKIP<INC1=038E             SKIP<INC2=0394             DEC<ADDRS=0395
SKIP<DEC1=039F             SKIP<DEC2=03A9             PRO.7F=03AA
PROC.DS=03B9               NO.PROC=03D1               /SAVER=8188
/ACCESS=8B86               /ID=A64E                   /MODE=00FD
/CONFIG=89A5               /ZERCK=832E                /P2SCR=829C
/LOADT=8C78                /NACCESS=8B9C              /RESXAF=81B8
USER/LOAD=03D5             SKPERRU/L=03F6             END.PGM=03F9

//0000,03F9,03F9
```

USER'S NOTES

150 South Wolfe Road
Sunnyvale, CA 94086