A 2K SYMBOLIC ASSEMBLER FOR THE 6502
A 2K SYMBOLIC ASSEMBLER FOR THE 6502

Robert Ford Denison
A 2K SYMBOLIC ASSEMBLER FOR THE 6502

Copyright 1979
by Robert Ford Denison
RD5 Teeter Rd.
Ithaca, NY 14850

All rights reserved, including the right to reproduce the program or documentation in machine-readable form, including magnetic media and read-only-memory.

Cover: Schematics for a 5V, 3A regulated power supply and a 1K x 8 read/write memory block. The power supply and three such memory blocks can be added to the basic KIM-1 microcomputer to provide the 4K RAM required by this assembler. Parts are available from Jameco Electronics.
# TABLE OF CONTENTS

1. INTRODUCTION ...................................................... 1
2. USE OF THE ASSEMBLER ........................................... 3
   2.1 Basic Concepts ............................................... 3
   2.2 Control Mode ............................................... 4
   2.3 Assembly Language Format .................................. 5
   2.4 Edit Mode Commands ......................................... 7
   2.5 Programming Restrictions .................................. 8
   2.6 Sample Run .................................................. 10
   2.7 Structured Programming .................................... 11
3. LISTING ........................................................................ 13
4. THEORY OF OPERATION ............................................. 41
   4.1 Encoding Scheme ............................................. 41
   4.2 Useful Subroutines ........................................... 41
5. MODIFICATION .......................................................... 46
   5.1 Changing Special Key Definitions ......................... 46
   5.2 Moving Tables ................................................. 46
   5.3 Adding Custom Commands ................................... 46
   5.4 Relocation .................................................... 47
   5.5 I/O Requirements ............................................. 47

APPENDIX A: AN INEXPENSIVE I/O SYSTEM ................. 49
APPENDIX B: ANSWERS TO USER QUESTIONS ............. 55

# TABLES

2.1 Input Format for Commands and Instructions ........ 6
2.2 Error Codes .................................................. 9
4.1 Important Arrays and Pointers ......................... 42
4.2 Global Symbols on Page Zero ......................... 43
4.3 Other Global Symbols ..................................... 44
4.4 Hierarchy of Modules ..................................... 45
5.1 I/O Routines ................................................ 48
1. INTRODUCTION

Microcomputers based on the powerful 6502 microprocessor are becoming increasingly widespread. Business, educational, and word-processing applications generally require expensive disk-based systems running high level languages such as BASIC or Pascal. Inexpensive 6502 systems have mainly been limited to such trivial uses as games, checkbook balancing, and recipe files. Games may, of course, be used for the nontrivial purpose of learning about microcomputers.

Inexpensive systems may, however, be more than adequate for quite sophisticated applications in the field of process control and data acquisition. A simple example is turning a tape recorder on at a specified time to record a radio program. Opening and closing insulated shutters to maximize solar heat gain while minimizing heat loss is more challenging, but could result in considerable savings. An example of a scientific application is collecting data from temperature and pressure sensors in a study of sap flow in sugar maples.

My own experience has been entirely with the MOS Technology KIM-1, which is ideal for such applications. I first used it to control an optical printer which was used to produce special cinematic effects. More recently, my KIM-1 was part of a complex gas analysis system for my research on nitrogen fixation in soybeans.

Neither expensive computer hardware nor years of training is necessary to attempt such projects. My system has only 4K RAM. I use a $30 software-scanned keyboard for input, and use the KIM-1 display as an output device for both numbers and letters. I learned most of what I know in this field from the MOS Technology Programming Manual, Don Lancaster's TTL Cookbook, BYTE magazine, and by trial and error.

The key to process control programming is the use of assembly language. It is much faster than BASIC, and uses far less memory than high level languages. In addition, most process control problems can be solved more easily and directly in assembly language than in a higher level language.

An assembler makes assembly language programming considerably easier by taking over the time-consuming and error-prone task of translating assembly language into machine language. A true assembler, such as the one described herein, allows the programmer to refer to variables, subroutines, and lines within subroutines using descriptive names, rather than their addresses.

This assembler outperforms all other true assemblers for the 6502 with which I am familiar, in terms of speed and memory efficiency. It can assemble a 128 byte module in a fraction
of a second. Programs up to 1K bytes can be assembled in a KIM-1 system with only 4K RAM, including 2K for the assembler itself. I would appreciate being informed of any other symbolic assembler which can match either of these claims.

I would like to thank Dr. H. R. Luxenberg, Professor of Computer Science at the California State University at Chico for modifying the assembler I/O for the SYM, and for pointing out errors in the program and documentation. John Geiger, of Milwaukee, found additional errors and kindly relocated the assembler to start at address 2000. Any errors that remain are my responsibility, and I would appreciate having them brought to my attention.

This book is dedicated to Mike Colyar, of the Evergreen State College, who introduced me to electronics.
2. USE OF THE ASSEMBLER

System requirements. The assembler requires a 650X-based microcomputer with at least 4K RAM and an appropriate I/O device. This documentation is based on a standard system: a KIM-1 with 3K RAM at address 0400 and a conventional computer terminal connected to the serial interface. A second version is available for KIM-1 systems with 4K RAM at address 2000; addresses in parentheses refer to that version.

Other systems. The assembler can be modified for use with other systems by following the guidelines in Section 5. More detailed instructions for specific systems will be made available as demand warrants. SYM owners see Appendix B.

Installing the assembler. To install the assembler in the standard system, load it from cassette or listing. Begin execution at address 05B8 (23B8). The assembler will prompt with a question mark, indicating that it is in control mode.

2.1 Basic Concepts

Modes. The assembler operates in two modes. "Control" mode allows control of the allocation of memory space, definition of variables, and related functions. "Edit" mode is used to actually enter, modify, and assemble modules.

Modules. A "module" is a subroutine or a segment of a program or subroutine. Each use of edit mode corresponds to one module. Modules are limited in length to 128 bytes, but a program may contain many modules. Total program length is limited only by available RAM.

Module pointer. Assembled modules are stored successively in RAM under the control of the "module pointer." This pointer is initialized to OC80 (2A80). It is then incremented automatically each time a module is stored, to prevent the module from being overwritten by the next module. More information on this and other pointers is given in Table 4.1.

Symbols. A "symbol" is a name given to a specific address. It may refer to a variable, a table, a module, a line within a module, or some other address such as an I/O port. Symbols may be up to six characters in length.

Global vs. local symbols. "Global" symbols are defined in control mode and may be referenced by any module. Symbols defined in edit mode are "local" to the module in which they were created and may not be referenced by other modules. Line labels are local symbols, so two modules may use identical line labels without confusion.
Input format. Input to the assembler must be in a specific format. Each input line is divided into a series of "fields." Each item must be left-justified in the correct field. In practice this is quite easy, because the "space" bar has been programmed to advance automatically to the beginning of the next field each time it is pressed. It may also be used to skip a field.

Special key definitions. Each line must be terminated with a carriage return. A "null line" consists of a carriage return only. "Backspace" may be used to correct errors within a given field; more serious errors require use of the assembler's editing capability. The "escape" key causes the assembler to execute a BRK instruction, and may be used to return to the system monitor. Users whose terminals lack any of the above keys should refer to Section 5.1.

Hexadecimal numbers. The assembler uses hexadecimal (base sixteen) numbers exclusively. All addresses in this documentation are therefore given in hexadecimal. Blanks are read as zeroes.

Arrays. An array is any variable, e.g. a table, that occupies more than one byte. Arrays are limited to 255 bytes. However, two or more arrays may be treated as one large array if an array longer than 255 bytes is needed.

Source vs. object code. "Source code" refers to the assembly language module. Assembly is the process of translating source into "object," or machine language code.

2.2 Control Mode

In this mode the user can define global symbols, allocate space for tables, redefine the module pointer, and enter edit mode to begin a new module. Control mode commands begin with a question mark, which is also a prompt symbol for the mode.

Enter the command in the first field, followed by any additional information required in subsequent fields. The format for each command is given in Table 2.1 and illustrated by example in Section 2.6.

Define global symbols. The ?ASSGN command is used to assign addresses to global symbols. A four-digit address is required. Additional symbols may be defined without typing "?ASSGN" again. Just hit the space bar to skip the first field; then enter the symbol and its address. Enter a null line (carriage return) when all symbols have been defined.

Allocate space for tables. Use the ?TABLE command to reserve space for tables. Enter the name of the table and its length in bytes (two digits). The symbol is assigned the
current value of the module pointer as its address. The pointer is then incremented by the length of the table to prevent over-write by the next table or module. Additional tables may be defined in a manner similar to that for ?ASSGN.

Redefine the module pointer. The ?REDEF command may be used with caution to change the value of the module pointer. This might be done to allow assembled modules to be stored in memory locations not ordinarily used for program storage. For example, assembled modules might be stored on page zero or one if space were at a premium.

Begin new module. The ?BEGIN command causes the assembler to enter edit mode to start a new module. The name of the module is entered in the second field, and is added to the symbol table as a global symbol. Its address is the current value of the module pointer, since that is where the module will be stored after it is assembled. The module name is also the label for the first line in the module, unless another line label is supplied.

2.3 Assembly Language Format

In edit mode, the user inputs an assembly language module. The module is edited and assembled using commands described in Section 2.4. This process is illustrated in Section 2.6. The prompt for edit mode is a hyphen, followed by the address where the assembly language code for the line will be stored.

To enter a line of assembly language, hit the space bar to skip over the first field. The contents of the other fields are summarized in Table 2.1 and further explained below.

Label. Enter a symbol in the second field if the line will be referenced by a branch instruction elsewhere in the module. Otherwise hit the space bar again.

Opcode. This field must contain the mnemonic and address mode for the desired instruction. The mnemonic is the standard three-letter MOS Technology code, e.g. LDA. Absolute, implied, and relative addressing require no additional information in this field. The other address modes are indicated in the opcode field by one or two characters immediately following the mnemonic, e.g. LDAZX. These mode codes are #,Z,A,IX,IY,ZX, X,Y,I, and ZY for immediate, zero page, accumulator, indexed indirect X, indirect indexed Y, zero page X, absolute X, absolute Y, indirect, and zero page Y addressing. Users who prefer IM for immediate addressing need only change two bytes at 02AC (20AC) to 49,4D.

Operand. For instructions that require no operand, hit carriage return to end the line. Immediate addressing requires a two-digit hexadecimal number in this field. Other address modes use a symbol as their operand.
<table>
<thead>
<tr>
<th>Table 2.1: Input Format for Commands and Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Field 1</strong></td>
</tr>
<tr>
<td>Assign address to symbol.</td>
</tr>
<tr>
<td>Reserve space for table.</td>
</tr>
<tr>
<td>Redefine module pointer.</td>
</tr>
<tr>
<td>Begin new module.</td>
</tr>
</tbody>
</table>

One-byte instructions.
Immediate mode instructions.
Other two-byte instructions.
Three-byte instructions.

Define local symbol. | -LOCAL | symbol | nnnn |
Assemble module. | -ASSEM |
Print lines in range. | -PRINT | nnTOnn |
Insert before line given. | -INSRT | nn |
Replace lines in range. | -INSRT | nnTOnn |
Append to end of module. | -INSRT | FF |
Save module in RAM. | -STORE |

() Optional.
nn Hexadecimal digits.
Offset. Three-byte instructions may use a two-digit hexadecimal number in this field to indicate an offset from the beginning of a table or array. This value is added to the base address of the array on assembly. The offset is optional, and may not be used with two-byte instructions.

2.4 Edit Mode Commands

Commands are used in edit mode to define local symbols and to assemble, list, edit, and save a module. Edit mode commands begin with a hyphen. Their format is given in Table 2.1 and their use is illustrated in Section 2.6.

Define local symbols. The -LOCAL command is identical to ?ASSGN except that the symbols defined are local to the module.

Assemble. The -ASSEM command translates the module into machine language. The assembler will respond quickly with either the normal address prompt, indicating successful assembly, or with one or more undefined symbols. Use the -LOCAL command to define these symbols before assembling again. Undefined global symbols may be temporarily defined locally to allow assembly.

List. An assembled module may be listed using the -PRINT command. Two line numbers must be supplied. The number of a line consists of the two least significant digits of its address prompt. -PRINT will list from the first line number up to, but not including, the second line number. The module must be reassembled before listing each time it is modified.

Test. The assembled module may be tested by hitting "reset" to return to the system monitor. Check the module pointer at 0040,41 to get the start address of the module. The module may be tested using appropriate user or monitor routines. Then return to edit mode by entering the assembler at 05D6 (23D6). Correct any errors (using the -INSRT command) and reassemble.

Insert lines. The -INSRT command can be used to insert, delete, or replace lines. To insert one or more lines, use -INSRT with a line number. New lines are inserted starting at that line number. The line previously at that address, and all lines following it, are automatically moved forward to make room for each new line.

Delete or replace lines. If a second line number is supplied with the -INSRT command, the assembler will delete the lines in the specified range. Lines following the deletion are moved back to fill the resulting gap. New lines can then be inserted starting at the first line number.
Append new lines. After inserting or deleting lines, the user may wish to add lines to the end of the module. To do this, type -INSRT FF (fast forward?). Ignore the resulting error code.

Save. An assembled module is saved using the -STORE command. The module length is added to the module pointer to prevent over-write by the next module. Memory space is conserved by clearing local symbols from the symbol table. The assembler then returns to control mode, allowing definition of new global symbols, redefinition of the module pointer, or beginning a new module.

Tape storage. Either source or object code can be saved on tape. Saving object code is easy since it only requires dumping the area of memory which contains the code itself. Saving source code requires saving both the symbol table and the module. This is done by dumping 0A00-0C7F (2800-2A7F). In addition, pointers at the following locations must be saved: 0030, 003D, 0050, 0051, 0056. It is probably easiest just to make a note of these pointer values, using the form at the end of this manual.

Retrieving modules from tape requires that the assembler be initialized by running it normally from 05B8 (23B8). Then hit "reset" to leave the assembler. Load the module from tape, restore the pointer values, and enter the assembler at 05D6 (23D6). Ignore any error message on re-entry.

Note that the previous contents of the symbol table are destroyed by this process, so that some global symbols may have to be redefined if the module is loaded for use with a new program. The assembled module will be stored according to the value of the module pointer before the module was loaded. This may not correspond to its previous location. ?REDEF may be used to store the assembled location wherever desired.

Saving and retrieving assembly language modules is a tricky process which requires experience to master. It may be easier to debug the module thoroughly and save the object code.

2.5 Programming Restrictions

The assembler is reasonably immune to user error, other than careless use of the ?REDEF command. Each input line is checked for correctness; when an error is detected, the normal prompt symbol is replaced with an error code (Table 2.2). The restrictions below are designed to eliminate errors at assembly time (other than undefined symbols) and to minimize debugging time.

Commands. Commands may be used at any time, but the result may be order-dependent. For example, ?TABLE will reserve space in a different place if used after ?REDEF. However, ?ASSGN uses absolute addresses and is unaffected by ?REDEF.
Module length. Module length is limited to 128 bytes. This guarantees that relative branches within a module will be within range. It also requires that programs be broken up into short modules which can be debugged more easily. A module listing will generally fit on one page. The length of a module corresponds to the two rightmost digits in the address prompt. Total program length is limited by available RAM.

Relative branches. Relative branches are allowed only within a module, for the reason given above. Line labels may only be referenced by relative branches; this greatly simplifies relocation.

Symbols. All symbols referenced in a module must be defined before assembly. This normally requires that subroutines be assembled and stored before they are referenced by a program or another subroutine. However, they could be assigned an address using ?ASSGN or -LOCAL, and entered later. Zero page symbols must be defined before the first line in which they are referenced.

Other restrictions. Symbol table length is limited to 64 symbols. No offset is permitted with two-byte instructions.

Table 2.2: Error Codes

A Command does not exist.
B Module length exceeds 128 bytes.
C Number of symbols exceeds 64.
D Symbol already defined.
K Command legal in edit mode only.

0 Command does not exist.
1 Mnemonic does not exist.
2 Address mode does not exist.
3 Illegal address mode for mnemonic.
4Operand undefined; must be on page zero.
5Operand not on page zero.
6 Offset legal for three-byte instructions only.
7 Relative branch illegal outside module.
8 Absolute addressing illegal within module.
9Command legal in control mode only.
%Illegal line number.
: Symbol already defined.
2.6 Sample Run

05B8 G
? TABLE WAVE OC
? TABLE
? ASSGN PAD 1700
? ASSGN PERIOD 0060
? ASSGN
? BEGIN DELAY
- 0C00 LDX# 2F
  1 0C00 LDX# 2F
- 0C02 DEX
- 0C03 BPL LOOP
- 0C05 RTS
- 0C06 -ASSEM
LOOP
- 0C06 -INSRT 02003
- 0C02 LOOP DEX
- 0C03 -ASSEM
- 0C06 -PRINT 00006
A22F DELAY LDX# 2F 00
CA LOOP DEX 02
10FD BPL LOOP 03
60 RTS 05
- 0C06 -STORE
? ? REDEF 0070
? ? BEGIN WAVGEN
- 0C00 -LOCAL BASE 0061
  -LOCAL
- 0C00 LDYZ PERIOD
- 0C02 LOOP LDAY WAVE
- 0C05 ADCIY BASE
- 0C07 STA PAD 02
- 0C0A JSR DELAY
- 0C0D DEY
- 0C0E BNE LOOP
- 0C10 RTS
- 0C11 -ASSEM
- 0C11 -PRINT 00011
A460 WAVGEN LDYZ PERIOD 00
B9800C LOOP LDAY WAVE 02
7161 ADCIY BASE 05
8D0217 STA PAD 02 07
208C0C JSR DELAY 0A
88 DEY 0D
D0F2 BNE LOOP 0E
60 RTS 10
- 0C11 -STORE
?
The array WAVE occupies the first twelve bytes of the program storage area. Thus, the module DELAY will begin at address 0C8C (2A8C).

Two global symbols were defined with a single use of the ?ASSGN command.

The assembler failed to recognize the opcode LDX# when it was entered in the wrong field.

The module could not be assembled at first because of the undefined symbol, LOOP. This was corrected using the -INSRT command to replace the unlabeled line.

The first line of a listing is labeled with the name of the module unless another label is given it.

The use of the ?REDEF command means that the module WAVGEN will begin at 0070.

Both LOOP and BASE are local symbols. The LOOP in one module will not be confused with that in the other, and BASE may not be referenced in another module.

The module WAVGEN may call DELAY as a subroutine since DELAY was entered first.

2.7 Structured Programming

The discipline of structured programming has become increasingly popular with the spread of such languages as Pascal. Structured programming in assembly language is more difficult, but offers the same advantages. Structured programs are more likely to run correctly the first time, easier to debug, and easier for other programmers to understand. Structured programming in machine language requires that the programmer accept the following restrictions on transfer of control.

**Blocks.** Every forward branch creates a block of one or more lines of assembly language, between the branch instruction and the line referenced by the branch. Execution of the block must begin with the first line of the block; no instruction outside the block may reference a line within the block. On completion of a block, control must pass to the line immediately following the block; no branch in the block may reference a line outside the block. Blocks may contain blocks and loops.

**Loops.** Every backward branch creates a loop. The loop includes the branch instruction and the line it references. The same restrictions given for blocks also apply to loops. Loops may contain loops and blocks.
Subroutines. Blocks and loops may contain subroutine calls. Since control returns to the calling block or loop, a subroutine may be considered as a nested block or loop.

Format. The structure of a module can be emphasized by indenting blocks and loops. This is illustrated throughout Section 3. Occasional NOP (EA) instructions were inserted to delimit blocks and loops. Nested loops or blocks may require two or three NOPs in a row, but rarely will an assembly language program contain a four EA series.
3. LISTING

Data Tables. MNETAB, MODTAB, etc.

0200 42 52 4B 43 4C 43 43 4C 44 43 4C 49 43 4C 56 44
0210 45 58 44 45 59 49 4E 58 49 4E 59 4E 4F 50 50 48
0220 41 50 48 50 50 4C 41 50 4C 50 52 54 49 52 54 53
0230 53 45 43 53 45 44 53 45 49 54 41 58 54 41 59 54
0240 53 58 54 58 41 54 58 53 54 59 41 43 50 58 53 54
0250 58 4C 44 58 43 50 59 4C 44 59 53 54 59 41 44 43
0260 41 4E 44 43 4D 50 45 4F 52 4C 44 41 4F 52 41 53
0270 42 43 53 54 41 41 53 4C 4C 53 52 52 4F 4C 52 4F
0280 52 44 45 43 49 4E 43 42 49 54 4A 4D 50 4A 53 52
0290 42 43 43 42 43 53 42 45 51 42 4D 49 42 4E 45 42
02A0 50 4C 42 56 43 42 56 53 20 20 41 20 23 20 5A 20
02B0 5A 58 5A 59 49 58 49 59 20 20 20 20 20 58 20 59 20
02C0 49 20 00 27 19 19 1D 1A 1F 1F 30 19 1D 1B 2E 19
02D0 2B 26 2E 2D 1C 27 27 38 30 2D 27 2F 00 F2 04 11
02E0 22 35 32 3A 31 50 63 75 6E 0C 80 0C A5 02 0E 00
02F0 03 02 37 C0 02 11 00 02 01 0C F8 09 15 00 08 05
0300 08 FF FF FF FF 00 18 D8 58 B8 CA 88 E8 C8 EA 48
0310 08 68 28 40 60 38 F8 78 AA A8 BA 8A 9A 98 0A 4A
0320 2A 6A E0 FF A2 C0 A0 FF 69 29 C9 49 A9 09 E9 E4
0330 86 A6 C4 A4 84 65 25 C5 45 A5 05 E5 85 06 46 26
0340 66 C6 E6 24 B4 94 75 35 D5 55 B5 15 F5 95 16 56
0350 36 76 D6 F6 B6 96 61 21 C1 41 A1 01 E1 81 71 31
0360 D1 51 B1 11 F1 91 90 B0 F0 30 D0 10 50 70 EC 8E
0370 AE CC AC 8C 6D 2D CD 4D AD 0D ED 8D 0E 4E 2E 6E
0380 CE EE 2C 4C 20 BC FF 7D 3D DD 5D BD 1D FD 9D 1E
0390 5E 3E 7E DE FE BE FF FF FF 79 59 D9 59 B9 19 F9
03A0 99 6C FF
03A3 Subroutine MATCH. Search table for match to reference, X points to search parameters on page zero. Sets z if match found, returns number of matching record in X.

| 86 29 | STXZ | ADL | Put address of search parameter list in ADL, H. |
| A2 00 | LDX# | 00  |
| 86 2A | STXZ | ADH |
| A0 06 | LDY# | 06  |
| B1 29 | PARAM | LDAIY | ADL |
| 99 30 00 | STAY | TBL |
| 88 | DEY |
| 10 F8 | BPL | PARAM |
| A6 36 | LDXZ | NUM |
| A4 35 | RECORD | LDYZ | HBC |
| B1 30 | BYTE | LDAIY | TBL |
| D1 32 | CMPY | RFL |
| F0 02 | BEQ | OK |
| A0 FF | LDY# | FF |
| 88 | DEY |
| 10 F5 | BPL | BYTE |
| C8 | INY |
| D0 01 | BNE | INCADR |
| 60 | RTS |
| 38 | INCADR | SEC |
| A5 30 | LDAZ | TBL |
| E5 34 | SBCZ | LEN |
| 85 30 | STAZ | TBL |
| B0 02 | BCS | DECNUM |
| C6 31 | DECZ | TBH |
| CA | DECNUM | DEX |
| 10 E1 | BPL | RECORD |
| 60 | RTS |

First Y+1 bytes must match. Mismatch. All ok? z set. Find base address of next record. Last record? z clear.
03D5 Subroutine HEX. Convert ASCII character pointed to by X to 4 binary bits in A.

B5 00  LDAZX  IOBUF Get character.
C9 40  CMP#  40 Number or letter?
30 03  BMI  NUMER Letter; adjust.
38  SEC
E9 07  SBC#  07
29 0F  NUMER AND#  0F Convert to binary.
60  RTS

03E1 Subroutine HX2BIN. Convert 2 ASCII characters on page zero, pointed to by X, to 8 binary bits in X.

20 D5 03  JSR  HEX Find high byte,
0A  ASLA
0A  ASLA
0A  ASLA
0A  ASLA
85 2D  STAZ  TEMP and low byte.
E8  INX
20 D5 03  JSR  HEX Combine.
05 2D  ORAZ  TEMP
AA  TAX
60  RTS
03F2 Subroutine BIN2HX. Convert 4 bits in A to an ASCII character. Store in page zero, X.

C9 0A  CMP# 0A  Number or letter?
30 03  BMI  NUMER  Letter; adjust.
18  69 07  ADD# 07
18  NUMER CLC  Convert to ASCII.
69 30  ADD# 30
95 00  STA# IOBUF  Store character.
60  RTS

03FF Subroutine DSPHEX. Convert binary number in A to two ASCII (hexadecimal) characters in page zero locations X, X+1.

48  PHA  Save number.
4A  LSRA  Find high character.
4A  LSRA
4A  LSRA
4A  LSRA
20 F2 03  JSR BIN2HEX  Find low character.
E8  INX
68  PLA
29 0F  AND# OF
20 F2 03  JSR BIN2HEX
60  RTS
040F Subroutine SYM. Puts base address of symbol table entry X in MISCL, H.

38 SEC
86 2D STXZ TEMP
A5 56 LDAZ SYMNUM
E5 2D SBCZ TEMP
85 2B STA Z MISCL
A9 00 LDA # 00
85 2C STA Z MISCH
18 CLC
A0 02 LDY # 02
26 2B X8 ROLZ MISCL
26 2C ROLZ MISCH
88 DEY
10 F9 BPL X8
38 SEC
A5 50 LDAZ SYMTBL
E5 2B SBCZ MISCL
85 2B STAZ MISCL
A5 51 LDAZ SYMTBH
E5 2C SBCZ MISCH
85 2C STA Z MISCH
60 RTS

0434 Subroutine ADDRSS. Puts address corresponding to symbol X in ADL, H.

20 0F 04 JSR SYM
A0 06 LDY # 06
B1 2B LDAIY MISCL
85 29 STA Z ADL
C8 INY
B1 2B LDAIY MISCL
85 2A STA Z ADH
60 RTS
0443 Subroutine ADDLAB. Add symbol to table. A points
to 6 zpage bytes containing symbol. Returns number of new
symbol in X.

85  29   STAZ    ADL
A9  00   LDA#    00   ADL,H points
to symbol.
85  2A   STAZ    ADH
18     CLC
A5  50   LDAZ    SYMTBL
69  08   ADC#    08
85  50   STAZ    SYMTBL
90  02   BCC     NOADDR
E6  51   INCZ    SYMTBH
A0  07   NOADDR  LDY#    07
A9  FF   LDA#    FF
91  50   STAIX   SYMTBL
88     DEY
88     DEY
B1  29   XFRSYM  LDAIX   ADL
91  50   STAIX   SYMTBL
88     DEY
10  F9   BPL     XFRSYM
A6  56   LDXZ    SYMNMD
E8     INX
86  56   STXZ    SYMNMD
60     RTS

0469 Subroutine NEWSYM. Puts base address of symbol table
record for symbol pointed to by A in MISCL, H and returns
symbol in X. If new, adds to table and sets Z.

85  52   STAZ    SYMRFL
A2  50   LDX#    50
20  A3  03  JSR     MATCH
F0  05   BEQ     OLD
A5  52   LDAZ    SYMRFL
20  43  04  JSR     ADDLAB
20  0F  04  OLD  JSR     SYM
E4  56   CPXZ    SYMNMD
60     RTS

Set up search.
Look up symbol.
Not found; add
to symbol table.
Address in MISCL, H.
Set z if new.
047D Subroutine ENCODE (part 1). Put mnemonic code in MNE, address mode in X.

A2 42 LDX# 42 Find mnemonic.
20 A3 03 JSR MATCH
F0 03 BEQ MNEFND
A9 31 LDA# 31 "1" Error-
60 RTS not found.
86 2E MNEFND STXZ MNE Save mnemonic.
A2 49 LDX# 49 Find address mode.
20 A3 03 JSR MATCH
F0 03 BEQ MODFND
A9 32 LDA# 32 "2" Error-
60 RTS not found.
A5 2E MODFND LDAZ MNE Special cases:
C9 19 CMP# 19 Implied mode.
10 02 BPL NOTIMP
A2 00 LDX# 00
C9 30 NOTIMP CMP# 30 Relative mode.
30 02 BMI NOTREL
A2 08 LDX# 08
EA NOTREL NOP

04A2 Subroutine ENCODE (part 2). Check legality of mnemonic/address mode combination.

A5 2E LDAZ MNE Legal mnemonic
DD C2 02 CMPX MIN for address mode?
10 03 BPL NOT2LO "3" Too low.
A9 33 LDA# 33
60 RTS
DD CF 02 NOT2LO CMPX MAX
30 03 BMI NOT2HI "3" Too high.
A9 33 LDA# 33
60 RTS
18 NOT2HI CLC
7D DC 02 ADCX BASE
85 37 STAZ OPCPTR Store pointer
AA TAX to opcode
BD 05 03 LDAX OPCTAB
C9 FF CMP# FF
D0 03 BNE OPCLGL
A9 33 LDA# 33 "3" Illegal.
60 RTS
EA OPCLGL NOP

19
04C6 Subroutine ENCODE (part 3). Find operand code, if required, for address modes other than relative and 3-byte address modes.

A5 37  LDAZ   OPCPTR
C9 1D   CMP#   1D
10 03   BPL    OPRRQD
A9 2D   LDA#   2D
60      RTS
E6 2F   OPRRQD INCZ  BYTES
C9 2A   CMP#   2A
10 0A   BPL    NOTIMM
A2 15   LDX#   15
20 A1 03 JSR    HX2BIN
86 38   STXZ   SYMPTR
A9 2D   LDA#   2D
60      RTS
A2 15   NOTIMM LDX#   15
86 52   STXZ   SYMRFL
C9 61   CMP#   61
10 20   BPL    NOTZPG
A2 50   LDX#   50
20 A3 03 JSR    MATCH
F0 03   BEQ    FOUND
A9 34   LDA#   34
60      RTS
20 34 04 FOUND JSR    ADDRSS
F0 03   BEQ    OK
A9 35   LDA#   35
60      RTS
86 38   OK    STXZ   SYMPTR
A5 1C   LDAZ   OFFSET
C9 20   CMP#   20
F0 03   BEQ    DONE
A9 36   LDA#   36
60      RTS
A9 2D   DONE  LDA#   2D
60      RTS
EA      NOTZPG  NOP

Consider opcode.
Operand required?
"-"
No; return.
At least 2 bytes.
Immediate addressing.
Find binary value
"-"
Set up operand search.
Zpage addressing?
Yes.
Look up operand.
"4" Not found.
"5" Not zpage.
Store operand.
Check for offset.
"SP"
"6" offset illegal.
OK, return.
Continue.
0508 Subroutine ENCODE (part 4). Look up operand; add if required.

A2 50   LDX# 50  Look up operand.
20 A3 03 JSR MATCH
F0 05   BEQ FOUND
A9 15   LDA# 15  Not found; add
20 43 04 JSR ADDLAB to symbol table.
86 38   FOUND STXZ SYMPTR
A5 37   LDAZ OPCPTR
C9 69   CMP# 69  Relative addressing?
10 0A   BPL NOTREL
E4 3C   CPXZ GLOBAL
10 03   BPL OK
A9 37   LDA# 37  "7" Error-
60     RTS branch not local.
A9 2D   OK LDA# 2D  
60     RTS 
EA     NOTREL NOP

0527 Subroutine ENCODE (part 5). For absolute addressing, check legality and find offset.

E4 3C   CPXZ GLOBAL  Operand must
30 0A   BMI OK be global or
20 34 04 JSR ADDRSS outside block.
C5 3F   CMPZ CRNTAH
D0 03   BNE OK
A9 38   LDA# 38  "8" Absolute
60     RTS mode w/in block.
A5 1C   OK LDAZ OFFSET
A2 00   LDX# 00  "SP"
C9 20   CMP# 20
F0 05   BEQ STROFS
A2 1C   LDX# 1C  Find offset.
20 E1 03 JSR HX2BIN
86 39   STROFS STXZ OPRDSP
E6 2F   INCZ BYTES
A9 2D   LDA# 2D  
60     RTS 

"_" Stay in edit mode.
0549 Subroutine CMAND. Look up and execute command.

A5 3A  LDAZ MODE Command legal
C5 00  CMPZ IOBUF for mode?
F0 04  BEQ OK
18    CLC No; illegal.
69 0C  ADC# 0C Return "9" or "K"
60    RTS
A9 00  OK LDA# 00 Look up command.
85 52  STAZ SYMRFL
A2 50  LDX# 50
20 A3 03 JSR MATCH
F0 0C  BEQ FOUND
A5 00  LDAZ IOBUF Not found.
C9 3F  CMP# 3F
10 03  BPL CMODE
A9 30  LDA# 30 "0" Error-
60    RTS input mode.
A9 41  CMODE LDA# 41 "A" Error-
60    RTS command mode.
A9 05  FOUND LDA# 05 Set up return.
48    PHA
A9 75  LDA# 75
48    PHA
20 34 04 JSR ADDRS Get address.
6C 29 00 JMPI ADL Execute command.
60    RTS
0577 Subroutine FIN. Add line to program; assign address to label, if any.

20 40 09 JSR INSERT Adjust if inserting.
A4 2F LDYZ BYTES
88 DEY
B9 37 00 ADDLIN LDAV OPCPTR Add line
91 3E STAIX CRNTAL to program.
88 DEY
10 F8 BPL ADDLIN
A5 07 LDAZ LABEL "SP"
C9 20 CMP# 20 Any label?
F0 10 BEQ INCADR Yes. Add to
A9 07 LDA# 07 symbol table
20 69 04 JSR NEWSYM if new, and
A0 07 LDY# 07 assign address.
A5 3F LDAZ CRNTAH
91 2B STAIX MISCL
88 DEY
A5 3E LDAZ CRNTAH
91 2B STAIX MISCL
18 INCADR CLC Increment pointers.
A5 3E LDAZ CRNTAH
65 2F ADCZ BYTES
85 3E STAZ CRNTAL
18 CLC
A5 3D LDAZ PRGLEN "B" Error-
65 2F ADCZ BYTES program overflow.
85 3D STAZ PRGLEN
10 03 BPL OK
A9 42 LDA# 42 "C" Error-
60 RTS symbol overflow.
24 56 OK BITZ SYMNUM
50 03 BVC OK2
A9 43 LDA# 43
60 RTS
A9 2D OK2 LDA# 2D
60 RTS
05B8 Main program. Process command, or translate input into source code.

D8  CLD
A2  18
BD E9 02 INIT LDAZ PRMTAB
95  3F STAZX CRNTAH
CA  DEX
10 F8 BPL INIT
A9  3F LDA#  3F "?" Set.
85  00 START STAZ IOBUF command mode.
A0  20 LDY#  20 "SP"
A2  21 LDX#  21
94  01 CLEAR STYZX IOBUFL Clear I/O buffer
CA  DEX
10 FB BPL CLEAR except error code.
A2  3F LDX#  3F "?" Command.
C9  3F CMP#  3F Command mode?
10  10 BPL GETLIN
A5  3F LDAZ CRNTAH No; input mode.
A2  02 LDX#  02 Display address.
20 FF 03 JSR DSPHEX
A5  3E LDAZ CRNTAL
A2  04 LDX#  04
20 FF 03 JSR DSPHEX
A2  2D LDX#  2D "-" Input.
86  3A GETLIN STXZ MODE Save mode.
A9  01 LDA#  01 Initialize.
85  2F STAZ BYTES
20  5D 07 JSR INPUT Input line.
A5  3A LDAZ MODE Mode?
C9  2D CMP#  2D "-"
D0  04 BNE CMODE Command mode?
A5  01 LDAZ IOBUFL Input mode command?
C9  20 CMP#  20 "SP"
D0  0C CMODE BNE EXEC If neither,
20  7D 04 JSR ENCODE translate line.
C9  2D CMP#  2D "-"
D0  03 BNE NG If line legal,
20  77 05 JSR FIN add to program.
A2  00 NG LDX#  00
F0  03 EXEC BEQ DONE If command,
20 49 05 JSR CMAND execute it.
18 DONE CLC
90 B6 BCC START Repeat until reset.
EA  NOP

24
0610 ? BEGIN. Add module name to symbol table; enter input mode.

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A9 07</td>
<td>LDA# 07</td>
<td></td>
<td>Add name to symbol table.</td>
</tr>
<tr>
<td>20 69 04</td>
<td>JSR NEWSYM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F0 03</td>
<td>BEQ OK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A9 44</td>
<td>LDA# 44</td>
<td></td>
<td>&quot;D&quot; Error-label in use.</td>
</tr>
<tr>
<td>60</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>86 3C</td>
<td>STXZ GLOBAL</td>
<td></td>
<td>Set local cutoff.</td>
</tr>
<tr>
<td>A9 00</td>
<td>LDA# 00</td>
<td></td>
<td>Clear pointers.</td>
</tr>
<tr>
<td>85 3E</td>
<td>STAZ CRNTAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85 3D</td>
<td>STAZ PRGLEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0 06</td>
<td>LDY# 06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91 2B</td>
<td>STAIY MISCL</td>
<td></td>
<td>Set start address =CRNTAL, H.</td>
</tr>
<tr>
<td>A5 3F</td>
<td>LDAZ CRNTAH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>INY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91 2B</td>
<td>STAIY MISCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A9 2D</td>
<td>LDA# 2D</td>
<td></td>
<td>&quot;-&quot; Set input mode.</td>
</tr>
<tr>
<td>60</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
062E ? ASSGN. Assign addresses to labels.

A5 07 LDAZ LABEL
C9 20 START CMP# 20 "SP"
D0 03 BNE MORE Label supplied?
A9 3F LDA# 3F No; done.
60 RTS
A9 07 MORE LDA# 07 Add symbol to table.
20 69 04 JSR NEWSYM "D" Error-
F0 03 BEQ NOTOLD label in use.
A9 44 LDA# 44 Assign address.
60 RTS
A2 0E NOTOLD LDX# 0E
20 E1 03 JSR HX2BIN
A0 07 LDY# 07
8A TXA
91 2B STAIX MISCL
A2 10 LDX# 10
20 E1 03 JSR HX2BIN
88 DEY
8A TXA
91 2B STAIX MISCL "SP"
A9 20 LDA# 20 clear I/O buffer
A2 0C LDX# 0C except prompt.
95 07 CLEAR STAZX LABEL
CA DEX
10 FB BPL CLEAR
20 5D 07 JSR INPUT Next symbol.
A5 07 LDAZ LABEL
10 CC BPL START
EA NOP
0665 -LOCAL. Add local symbols to symbol table; assign addresses.

20 2E 06  JSR ?ASSGN  Add to symbol table
C9 44  CMP# 44  if new.
D0 03  BNE OK  ":" Error-
A9 3A  LDA# 3A  symbol in use.
60  RTS  "-" stay in input mode.
A9 2D  OK  LDA# 2D
60  RTS

0672 ?REDEF. Redefine module start address.

A2 07  LDX# 07  Find high address.
20 E1 03  JSR HX2BIN  Store.
86 41  STXZ MDLADH  Find low address.
A2 09  LDX# 09  Store.
20 E1 03  JSR HX2BIN  "?" stay in command mode.
86 40  STXZ MDLADL
A9 3F  LDA# 3F
60  RTS
0683 Subroutine ASMBL. Translate line into machine code; store result at (OBJECT). Return length-1 in Y.

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0 00</td>
<td>LDA# 00</td>
<td>Get first byte.</td>
<td></td>
</tr>
<tr>
<td>B1 3E</td>
<td>LDAIY CRNTAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>TAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BD 05 03</td>
<td>LDAX OPCTAB</td>
<td>Look up opcode.</td>
<td></td>
</tr>
<tr>
<td>91 57</td>
<td>STAIY OBJECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E0 1D</td>
<td>CPX# 1D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 01</td>
<td>BPL OPREQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>RTS</td>
<td>No operand.</td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>OPREQ INY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B1 3E</td>
<td>LDAIY CRNTAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E0 2A</td>
<td>CPX# 2A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 03</td>
<td>BPL NOTIMM</td>
<td>Address mode?</td>
<td></td>
</tr>
<tr>
<td>91 57</td>
<td>STAIY OBJECT</td>
<td>Immediate.</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>86 2E</td>
<td>NOTIMM STXZ MNE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA</td>
<td>TAX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 34 04</td>
<td>JSR ADDRS</td>
<td>Get address.</td>
<td></td>
</tr>
<tr>
<td>A5 29</td>
<td>LDAZ ADL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0 01</td>
<td>LDY# 01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6 2E</td>
<td>LDXZ MNE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E0 61</td>
<td>CPX# 61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 03</td>
<td>BPL NOTZPG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91 57</td>
<td>STAIY OBJECT</td>
<td>Zero page.</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E0 69</td>
<td>NOTZPG CPX# 69</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 09</td>
<td>BPL NOTREL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>SEC</td>
<td>Relative.</td>
<td></td>
</tr>
<tr>
<td>E9 02</td>
<td>SBC# 02</td>
<td>Compute branch.</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>SEC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E5 3E</td>
<td>SBCZ CRNTAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91 57</td>
<td>STAIY OBJECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>NOTREL CLC</td>
<td>Absolute.</td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>INY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>71 3E</td>
<td>ADCIY CRNTAL</td>
<td>Add offset.</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>DEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91 57</td>
<td>STAIY OBJECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>INY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5 2A</td>
<td>LDAZ ADH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>69 00</td>
<td>ADC# 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91 57</td>
<td>STAIY OBJECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
06CB  Subroutine LOCSYM. Displays undefined local symbols.

A6  3C  LDXZ  GLOBAL  For local symbols,
E8    NXTSYM  INX
20  34  04  JSR  ADDRSH  see if defined.
C9    FF  CMP#  FF
D0    11  BNE  DEFIND  If not,
A0    05  LDY#  05  display symbol.
B1  2B  SHOW  LDAIY  MISCL
99  00  00  STAY  IOBUF
88    DEY
10  F8  BPL  SHOW
86  2B  STXZ  MISCL
20  A1  08  JSR  OUTLIN
A6  2B  LDXZ  MISCL
E4  56  DEFIND  CPXZ  SYMNUM  If more
30  E3  BMI  NXTSYM  symbols, repeat.
60    RTS

06EB  -ASSEM. Assemble module; store result in RAM
locations beginning at (MDLADR, H).

20  CB  06  JSR  LOCSYM  Check for local
A9  2D  LDA#  2D  undefined symbols.
C5  00  CMPZ  IOBUF
F0  01  BBIQ  ALLOC
60    RTS
A9  00  ALLOC  LDA#  00  Else, assemble.
85  3E  STAZ  CRNTAL  Initialize pointers.
A5  40  LDAZ  MDLADR
85  57  STAZ  OBJECT
A5  41  LDAZ  MDLADH
85  58  STAZ  OBJECT
20  83  06  NEXTLN  JSR  ASMBL  Translate a line.
84  2D  STYZ  TEMP  Save bytes -1.
38    SEC
A5  57  LDAZ  OBJECT  Increment pointers.
65  2D  ADCZ  TEMP  For object code.
85  57  STAZ  OBJECT
90  02  BCC  SKIP
E6  58  INCZ  OBJECT 1
38    SKIP  SEC  For source code.
A5  3E  LDAZ  CRNTAL
65  2D  ADCZ  TEMP
85  3E  STAZ  CRNTAL
C5  3D  CMPZ  PRGLEN
30  E5  BMI  NEXTLN  Finished?
A9  2D  LDA#  2D  "-"  Stay in
60    RTS  edit mode.
071F  ? TABLE. Allocate space for tables.

A5 07  LDAZ  LABEL
C9 20  START  CMP#  20  "SP"
D0 03  BNE  MORE  Any label?
A9 3F  LDA#  3F  No, done.
60  RTS
A9 07  MORE  LDA#  07
20 69 04  JSR  NEWSYM  Add symbol to
F0 03  BEQ  NOTOLD  symbol table.
A9 44  LDA#  44  "D" Error-
60  RTS  not new.
A0 06  NOTOLD  LDY#  06  Assign address.
A5 40  LDAZ  MDLADL
91 2B  STAIY  MISCL
C8  INY
A5 41  LDAZ  MDLADH
91 2B  STAIY  MISCL
A2 0E  LDX#  0E  Allocate space
20 E1 03  JSR  HX2BIN  by incrementing
8A  TXA  MDLADL, H.
18  CIC
65 40  ADCZ  MDLADL
85 40  STAZ  MDLADL
90 02  BCC  NOINC
E6 41  INCZ  MDLADH
A9 20  NOINC  LDA#  20  "SP"
A2 0C  LDX#  0C
95 07  CLEAR  STAZX  LABEL  Clear I/O buffer
CA  DEX  except prompt.
10 FB  BPL  CLEAR 20 5D 07  JSR  INPUT
A5 07  LDAZ  LABEL  Another symbol?
10 C5  BPL  START
EA  NOP
075D  Subroutine INPUT. Prompt w/ first word in IOBUF.  
Input up to 5 words. Special keys: ESC, CR, BKSP, SP.

20  2F  LE     JSR     CRLF
A2  00     LDX#     00     New line.  
B5  00     PROMPT   LDAZX   IOBUF
20  A0  LE     JSR     OUTCH
E8     INX
E0  06     CPX#     06
30  F6     BMI     PROMPT
A2  00     LDX#     00     Prompt w/  
A9  06     LDA#     06     first 6 chars.  
85  2D     STAZ    TEMP    includes space.  
20  5A  LE     START   JSR     GETCH    Input a char.  
C9  1B     CMP#     0B     "ESC"
D0  01     BNE     NOTBRK
00     BRK     Break.  
C9  0D     NOTBRK  CMP#     0D     "CR"
D0  01     BNE     NOTCR
60     RTS     End of line.  
C9  08     NOTCR   CMP#     08     "BS"
D0  05     BNE     NOTBSP
CA     DEX     Backspace.  
E6  2D     INCZ    TEMP
A9  08     LDA#     08
C9  20     NOTBSP  CMP#     20     "SP"
D0  0D     BNE     NOTSP
EA     NOP     Next word.  
20  9E  LE     TAB     JSR     OUTSP    Add spaces  
E8     INX     to fill word.  
C6  2D     DECZ    TEMP
10  F8     BPL     TAB
A9  06     LDA#     06
85  2D     STAZ    TEMP
C9  20     NOTSP  CMP#     20     If not a  
30  05     BMI     DONE    control char:  
95  00     STAZX   IOBUF    Add char to  
E8     INX     I/O buffer.  
C6  2D     DECZ    TEMP
18     DONE   CLC
90  CD     BCC     START    Next character.  
EA     NOP
07A6  -STORE. Clear local symbols; assign address to module.
Increment MDLADL,H to prevent overwrite by next module.
Return to command mode.

A6 3C   LDXZ  GLOBAL  Clear local
20 0F 04 JSR SYM symbols from
86 56   STXZ SYMNUM symbol table.
A5 2B   LDAZ MISCL
85 50   STAZ SYMTBL
A5 2C   LDAZ MISCH
85 51   STAZ SYMTBH
A0 07   LDY# 07 Assign address
A5 41   LDAZ MDLADH to module.
91 2B   STAIZ MISCL
88     DEY
A5 40   LDAZ MDLADL
91 2B   STAIZ MISCL
18     CLC
65 3D   ADCZ PRGLEN Increment MDLADL,H
85 40   STAZ MDLADL by length of
90 02   BCC SKIP module.
E6 41   INCZ MDLADH
A9 3F   LDA# 3F "?" Return to
60     RTS command mode.
Table MODLIM. Lower opcode pointer limits for modes.

07CC  00 19 1D  2A 3F 4F  51 59 61 69 80 90 9C

07D9 Subroutine DECODE. Decode line pointed to by CRNTAL and OBJECT. Put line in IOBUF, length in BYTES.

A9 01  LDA#  01  Assume 1 byte.
85 2F  STAZ  BYTES
A2 22  LDX#  22  Clear I/O buffer.
A9 20  LDA#  20
95 00  CLEAR  STAZX  IOBUF
CA  BPL  CLEAR
10 FB  LDXZ  SYMNUM  Check for label.
A6 56  JSR  ADDRSS  Compare address
to current line.
20 34 04  START  LDAZ  CRNTAL
A5 3E  CMPZ  ADL
C5 29  BNE  SKIP
D0 04  LDAZ  CRNTAH
A5 3F  CMPZ  ADH
C5 2A  BNE  SKIP2  If they match,
D0 0C  SKIP  LDY#  05  put label in
A0 05  LDAIY  MISCL  I/O buffer.
B1 2B  DEY  STAY  LABEL
LABL
99 07 00  LDX#  01  End search.
88  BPL  LABL
10 F8  E4 3C  DEX  Consider local
10 E0  CPFXZ  GLOBAL  symbols only.
A0 00  START
B1 57  LDX#  00  Get opcode.
A2 00  LDAIY  OBJECT
20 FF 03  JSR  DSPHEX
B1 3E  LDAIY  CRNTAL
85 37  STAZ  OPCPTR

33
Subroutine DECODE (part 2). Decode address mode and opcode; put in I/O buffer.

A2 0C  LDX#  0C  Find mode.
C9 1D  CMP#  1D  Any operand?
10 02  BPL    FNDMOD If not, only check
A2 01  LDX#  01  implied and accum.
DD CC 07  FNDMOD CMPX MODLIM In range
30 04  BMI   NOPE  for mode?
86 3A  STXZ  MODE  Yes; save mode.
A2 00  LDX#  00  End search.
CA NOPE DEX
10 F4  BPL    FNDMOD Put mode in
A5 3A  LDAZ  MODE  I/O buffer.
0A    ASLA
AA    TAX
BD A8 02  LDAX  MODTAB
85 11  STAZ  OPCOD3
BD A9 02  LDAX  MODTAB 01
85 12  STAZ  OPCOD4
B1 3E  LDAY  CRNTAL Find mnemonic.
38    SEC
A6 3A  LDXX  MODE Mnemonic number.
FD DC 02  SBCX  BASE Multiply by 3.
85 2D  STAZ  TEMP
0A    ASLA
18    CLC
65 2D  ADCZ  TEMP Get ASCII.
AA    TAX Put mnemonic in
BD 00 02  LDAX  MNETAB I/O buffer.
85 0E  STAZ  OPCODE
BD 01 02  LDAX  MNETAB 01
85 0F  STAZ  OPCOD1
BD 02 02  LDAX  MNETAB 02
85 10  STAZ  OPCOD2
A5 37  LDAZ  OPCPTR Operand needed?
C9 1D  CMP#  1D
10 01  BPL   OPRND
60    RTS No; finished.
E6 2F  OPRND INCZ  At least 2 bytes.
085E Subroutine DECODE (part 3). Decode operands and offset, if any.

A0 01  
B1 57  
A2 02  
20 FF 03  
A5 37  
C9 2A  
10 08  
B1 3E  
A2 15  
20 FF 03  
60  
B1 3E  
AA  
20 0F 04  
A0 05  
B1 2B  
99 15 00  
88  
10 F8  
A5 37  
C9 69  
10 01  
60  
E6 2F  
A0 02  
B1 57  
A2 04  
20 FF 03  
B1 3E  
P0 05  
A2 1C  
20 FF 03  
60  

08A1 Subroutine OUTLIN. Output line from IOBUF.

20 2F 1E  
A2 00  
B5 00  
20 A0 1E  
E8  
E0 23  
30 F6  
60  


New line. Output one character at a time, until done.
08B1 Subroutine PRNTCK. Check that FIRST and LAST are legal
line numbers. Print lines in range if PRNTOK=1.

LDA# 00  
STA CRNTAL
LDA MDLADL
STA OBJECT
LDA MDLADH
STA OBJECT1
LDA# 07
JSR HX2BIN
STXZ FIRST
LDX# 0B
JSR HX2BIN
STXZ LAST
LDA# 02
STA WRONG
JSR DECODE
LDA CRNTAL
CMPZ FIRST
BNE SKIP
DECZ WRONG
CMPZ LAST
BNE SKIP2
DECZ WRONG
CMPZ FIRST
BPL HIGH
BITZ PRNTOK
BMI NOPRNT
LDX# 1F
JSR DSPHEX
JSR OUTLIN
Print line.

NOPRNT
NOP
CLC
LDA OBJECT
ADcz BYTES
STA OBJECT
BCC NOINC
INcz OBJECT1

NOINC
CLC
LDA CRNTAL
ADcz BYTES
STA CRNTAL
CMPZ PRGLEN
BMI NXTLIN

RTS

Initialize.
Decode range.
Initialize flag for mismatch.
Decode line.
Decrement WRONG each time a match is found.
In range for print?
Yes, but print wanted?
Yes; add
Last line?
If not, repeat.
090D  PRINT. Output lines in specified range.

A9 01  LDA# 01  Set print flag.
85 38  STAZ PRNTOK
20 B1 08 JSR PRNTCK  Run print routine.
A9 2D  LDA# 2D  "-" Stay in
       RTS  edit mode.

0917 Subroutine FIXSYM. Adds BYTES to addresses of line labels. Used by -INSRT and subroutine INSERT.

A6 56  LDXZ SYMNUM  For local symbols,
20 34 04 START JSR ADDRSS  find address.
C5 3F  CMPZ CRNTAH  Line label?
D0 1A  BNE NOTLAB
A5 29  LDAZ ADL  Yes, but in
C5 3E  CMPZ CRNTAL  move zone?
30 13  BMI NOREV
A4 29  LDYZ ADL  Yes.
C4 5A  CPYZ LAST  Line deleted?
10 06  BPL NEWADR
A9 FE  LDA# FE  Yes.
A0 07  LDY# 07  Delete symbol.
91 2B  STAIY MISCL
18  NEWADR CLC  Fix address
65 2F  ADCZ BYTES
A0 06  LDY# 06
91 2B  STAIY MISCL
EA  NOREV NOP
CA  NOTLAB DEX  More local
E4 3C  CPXZ GLOBAL symbols?
10 DA  BPL START
60  RTS
0940 Subroutine INSERT. Open gap in program to insert current line. Adjust symbol table.

<table>
<thead>
<tr>
<th>A5 3E</th>
<th>LDAZ</th>
<th>CRNTAL</th>
<th>Inserting line?</th>
</tr>
</thead>
<tbody>
<tr>
<td>C5 3D</td>
<td>CMPZ</td>
<td>PRGLEN</td>
<td></td>
</tr>
<tr>
<td>D0 01</td>
<td>BNE</td>
<td>INS</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85 5A</td>
<td>INS</td>
<td>STAZ</td>
<td>Nope.</td>
</tr>
<tr>
<td>20 17 09</td>
<td>JSR</td>
<td>FIXSYM</td>
<td>Fix symbols.</td>
</tr>
<tr>
<td>18</td>
<td>CLC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5 3E</td>
<td>LDAZ</td>
<td>CRNTAL</td>
<td>Set up offset pointer for move.</td>
</tr>
<tr>
<td>65 2F</td>
<td>ADCZ</td>
<td>BYTES</td>
<td></td>
</tr>
<tr>
<td>85 29</td>
<td>STAZ</td>
<td>ADL</td>
<td></td>
</tr>
<tr>
<td>A5 3F</td>
<td>LDAZ</td>
<td>CRNTAH</td>
<td></td>
</tr>
<tr>
<td>85 2A</td>
<td>STAZ</td>
<td>ADH</td>
<td></td>
</tr>
<tr>
<td>A5 3D</td>
<td>LDAZ</td>
<td>PRGLEN</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>SEC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E5 3E</td>
<td>SBCZ</td>
<td>CRNTAL</td>
<td></td>
</tr>
<tr>
<td>A8</td>
<td>TAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B1 3E</td>
<td>MOVE</td>
<td>LDAIY</td>
<td>Move lines to open gap.</td>
</tr>
<tr>
<td>91 29</td>
<td>STAIY</td>
<td>ADL</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>DEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 F9</td>
<td>BPL</td>
<td>MOVE</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>RTS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

38
0965 -INSRT. Check supplied line numbers for legality. Set program pointer to first line number; delete to second.

A9 FF     LDA# FF  Legal line?
85 38    STAZ PRNTOK
20 B1 08  JSR PRNTCK
C5 5A     CMPZ LAST  Last+1 is legal line number.
D0 02     BNE NOTLST
C6 39     DECB WRONG
A5 39     NOTLST LDAZ WRONG
F0 03     BEQ OK
A9 25     LDA# 25  "&" Error-illegal address.
60 LDAZ FIRST
A5 59     OK RTS
85 3E     STAZ CRNTAL
A6 5A     LDXZ LAST
F0 26     BEQ DONE
38 SEC
E5 5A     SBCZ LAST
85 2F     STAZ BYTES
20 17 09  JSR FIXSYM
A5 3F     LDAZ CRNTAH
85 5B     STAZ LAST1
A5 3D     LDAZ PRGLEN
38 SEC
E5 3E     SBCZ CRNTAL
85 2D     STAZ TEMP
A5 3D     LDAZ PRGLEN
18 CLC
65 2F     ADCZ BYTES
85 3D     STAZ PRGLEN
A0 00     LDY# 00  Move lines to close gap.
B1 5A     MOVE LDAIY LAST
91 3E     STAIY CRNTAL
C8 INY
C4 2D     CPYZ TEMP
30 F7     BMI MOVE
EA NOP
A9 2D     DONE LDA# 2D  "-" Stay in edit mode.
60 RTS

09AA Move first nine entries in symbol table to RAM. Entry point for assembler in ROM.

A2 47     LDX# 47
BD B8 09  MOVSYM LDAZ ROM
9D B8 09  STAX RAM
CA DEX
10 F7     BPL MOVSYM
4C B8 05  JMP MAIN

39
<table>
<thead>
<tr>
<th></th>
<th>09B8</th>
<th>3F 41 53 53 47 4E 2E 06</th>
</tr>
</thead>
<tbody>
<tr>
<td>09C0</td>
<td>09C8</td>
<td>2D 4C 4F 43 41 4C 65 06</td>
</tr>
<tr>
<td>09D0</td>
<td>09D8</td>
<td>2D 41 53 53 45 4D EB 06</td>
</tr>
<tr>
<td>09E0</td>
<td>09E8</td>
<td>2D 53 54 4F 52 45 A6 07</td>
</tr>
<tr>
<td>09F0</td>
<td>09F8</td>
<td>2D 49 4E 53 52 54 65 09</td>
</tr>
</tbody>
</table>
4. THEORY OF OPERATION

4.1 Encoding Scheme

The assembler owes its speed and memory efficiency to the encoding scheme by which each line of assembly language is stored. As each line is entered, it is translated into an encoded form which is the same length as its machine language equivalent. This is done by Subroutine ENCODE. The result may be seen at the address given in the prompt for each line.

Opcode. The first byte in the coded assembly language for a line is a pointer to the opcode for the instruction. The opcodes are found in OPCTAB, but in an unusual order. They are grouped by address mode, with the address modes in the order given in Section 2.3. This arrangement simplifies coding, since the modes are arranged in order of number of bytes required. The mnemonics have also been rearranged, to eliminate gaps in the table.

Operand. For two- and three-byte instructions, the second byte in the assembly code is for the operand. This is just a hexadecimal number for immediate addressing. For the other address modes, it is the number of the symbol table entry for the operand. Each symbol table entry is eight bytes—six ASCII characters followed by the low and high address for the symbol. Hexadecimal FF for the high address indicates that no address has yet been assigned to the symbol.

Offset. For three-byte instructions, the third byte in the assembly code is the offset described in Section 2.3. This will be zero unless an offset is supplied.

Listing. When the -PRINT command is used, the encoded assembly language must be translated back into strings of ASCII characters. This is done by Subroutine DECODE.

Assembly. With this encoding scheme, final assembly is reduced to one or two table look-ups for each line. Most of the work is done during the carriage return time as each line is entered.

4.2 Useful Subroutines

Some of the subroutines in the assembler may be of use in user programs. HX2BIN and DSPHEX are examples. Subroutine MATCH is a powerful string-search routine. It requires the following information from the calling routine: base address of the last record in the table to be searched, start address of the string to be compared, record length for the table, number of the highest byte which must match (the record may contain additional information), and the number of the last record in the table. This information is passed in the form
of a single byte in the X register, which points to a page-zero array of these parameters. These correspond to the symbols TBL through NUM in Table 4.2. X is also used to return the number of the record which matches the supplied string. The zero flag is cleared if no match is found.

Table 4.1: Important Arrays and Pointers.

<table>
<thead>
<tr>
<th>Array</th>
<th>Assembly language module</th>
<th>Assembled program</th>
<th>Symbol table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>0C00-0C7F</td>
<td>0C80- ??</td>
<td>09B8-0BB7</td>
</tr>
<tr>
<td>range</td>
<td>(2A00-2A7F)</td>
<td>(2A80- ??)</td>
<td>(27B8-29B7)</td>
</tr>
<tr>
<td>Pointer</td>
<td>CRNTAL,H</td>
<td>MDLADL,H</td>
<td>SYMTBL,H</td>
</tr>
<tr>
<td></td>
<td>003E,003F</td>
<td>0040,0041</td>
<td>0050,0051</td>
</tr>
<tr>
<td>Points to</td>
<td>current line</td>
<td>first line</td>
<td>latest</td>
</tr>
<tr>
<td>Initial</td>
<td>current line</td>
<td>first line</td>
<td>latest</td>
</tr>
<tr>
<td>value</td>
<td>of module</td>
<td>of module</td>
<td>symbol</td>
</tr>
<tr>
<td>Initialized</td>
<td>0C00</td>
<td>0C80</td>
<td>09F8*</td>
</tr>
<tr>
<td>from</td>
<td>(2A00)</td>
<td>(2A80)</td>
<td>(27F8)</td>
</tr>
<tr>
<td></td>
<td>02E9**</td>
<td>02EA,02EB</td>
<td>02FA,02FB</td>
</tr>
<tr>
<td></td>
<td>(20E9)</td>
<td>(20EA,20EB)</td>
<td>(20FA,20FB)</td>
</tr>
</tbody>
</table>

?? Limited by available RAM.
() Address for version beginning at 2000.
* First part of symbol table reserved by assembler.
** High order address; low order initialized to zero.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOBUF</td>
<td>0000</td>
<td>I/O buffer; prompt or command field.</td>
</tr>
<tr>
<td>LABEL</td>
<td>0007</td>
<td>I/O buffer; label field.</td>
</tr>
<tr>
<td>OPCODE</td>
<td>000E</td>
<td>I/O buffer; opcode field.</td>
</tr>
<tr>
<td>Oprand</td>
<td>0015</td>
<td>I/O buffer; operand field.</td>
</tr>
<tr>
<td>USER</td>
<td>0023</td>
<td>Six bytes available for use by user commands.</td>
</tr>
<tr>
<td>ADL</td>
<td>0029</td>
<td>Low address pointer for various subroutines.</td>
</tr>
<tr>
<td>ADH</td>
<td>002A</td>
<td>High address pointer.</td>
</tr>
<tr>
<td>MISCL</td>
<td>002B</td>
<td>Miscellaneous uses.</td>
</tr>
<tr>
<td>MISCH</td>
<td>002C</td>
<td>Ditto.</td>
</tr>
<tr>
<td>TEMP</td>
<td>002D</td>
<td>Various temporary uses.</td>
</tr>
<tr>
<td>MNE</td>
<td>002E</td>
<td>Mnemonic code.</td>
</tr>
<tr>
<td>BYTES</td>
<td>002F</td>
<td>Lengths of lines, etc.</td>
</tr>
<tr>
<td>TBL</td>
<td>0030</td>
<td>Low address pointer for table; used by MATCH.</td>
</tr>
<tr>
<td>TBH</td>
<td>0031</td>
<td>High address pointer (Subroutine MATCH).</td>
</tr>
<tr>
<td>RFL</td>
<td>0032</td>
<td>Low address pointer for string to be matched.</td>
</tr>
<tr>
<td>RPH</td>
<td>0033</td>
<td>High address pointer (MATCH).</td>
</tr>
<tr>
<td>LEN</td>
<td>0034</td>
<td>Length of each record in table (MATCH).</td>
</tr>
<tr>
<td>HBC</td>
<td>0035</td>
<td>Number of highest byte in record which must match.</td>
</tr>
<tr>
<td>NUM</td>
<td>0036</td>
<td>Number of highest record in table (MATCH).</td>
</tr>
<tr>
<td>OPCPTR</td>
<td>0037</td>
<td>Pointer to opcode in OPCTAB.</td>
</tr>
<tr>
<td>PRNTCK</td>
<td>0038</td>
<td>Flag to enable printing by Subroutine PRNTCK.</td>
</tr>
<tr>
<td>WRRNG</td>
<td>0039</td>
<td>Flag for illegal line numbers (PRNTCK).</td>
</tr>
<tr>
<td>MODE</td>
<td>003A</td>
<td>Code for address mode.</td>
</tr>
<tr>
<td>SAVX</td>
<td>003B</td>
<td>Used to preserve X register.</td>
</tr>
<tr>
<td>GLOBAL</td>
<td>003C</td>
<td>Number of last global symbol.</td>
</tr>
<tr>
<td>PRGLEN</td>
<td>003D</td>
<td>Length of source code.</td>
</tr>
<tr>
<td>CRNTAL</td>
<td>003E</td>
<td>Low address pointer to current source code line.</td>
</tr>
<tr>
<td>CRNTAH</td>
<td>003F</td>
<td>High address pointer.</td>
</tr>
<tr>
<td>MDLADL</td>
<td>0040</td>
<td>Module pointer, low address.</td>
</tr>
<tr>
<td>MLDADH</td>
<td>0041</td>
<td>Module pointer, high address.</td>
</tr>
<tr>
<td>MNENTBL</td>
<td>0042</td>
<td>Parameters for MNENTAB (see TBL to NUM above).</td>
</tr>
<tr>
<td>MODTB1</td>
<td>0049</td>
<td>Parameters for MODTAB.</td>
</tr>
<tr>
<td>SYMTBL</td>
<td>0050</td>
<td>Low address pointer to last entry in symbol table.</td>
</tr>
<tr>
<td>SYMTBH</td>
<td>0051</td>
<td>High address pointer.</td>
</tr>
<tr>
<td>SYMRFL</td>
<td>0052</td>
<td>Low address pointer for symbol to be compared.</td>
</tr>
<tr>
<td>SYMRFH</td>
<td>0053</td>
<td>High address pointer.</td>
</tr>
<tr>
<td>SYMNUM</td>
<td>0056</td>
<td>Number of last symbol.</td>
</tr>
<tr>
<td>OBJECT</td>
<td>0057</td>
<td>Low address pointer to object code.</td>
</tr>
<tr>
<td>OBJCT1</td>
<td>0058</td>
<td>High address pointer.</td>
</tr>
<tr>
<td>FIRST</td>
<td>0059</td>
<td>First line in range for print (PRNTCK).</td>
</tr>
<tr>
<td>LAST</td>
<td>005A</td>
<td>First line after print range.</td>
</tr>
<tr>
<td>LAST1</td>
<td>005B</td>
<td>High order address; same as CRNTAH.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>*MNETAB 0200</td>
<td>Three-character ASCII mnemonics for instructions.</td>
<td></td>
</tr>
<tr>
<td>*MODTAB 02A8</td>
<td>Two-character ASCII mode codes.</td>
<td></td>
</tr>
<tr>
<td>*MIN 02C2</td>
<td>Minimum legal value for MNE for each mode.</td>
<td></td>
</tr>
<tr>
<td>*MAX 02CF</td>
<td>Lowest illegal value of MNE for each mode.</td>
<td></td>
</tr>
<tr>
<td>*BASE 02DC</td>
<td>Base value for mode added to MNE to get OPCPTR.</td>
<td></td>
</tr>
<tr>
<td>*PRMTAB 02E9</td>
<td>Initialization values for CRNTAH through SYMNUM.</td>
<td></td>
</tr>
<tr>
<td>*USRPRM 0301</td>
<td>Four bytes available for user parameters.</td>
<td></td>
</tr>
<tr>
<td>*OPCTAB 0305</td>
<td>Machine language opcodes pointed to by OPCPTR.</td>
<td></td>
</tr>
<tr>
<td>MATCH 03A3</td>
<td>Search table for match to reference.</td>
<td></td>
</tr>
<tr>
<td>HEX 03D5</td>
<td>ASCII character to four bits.</td>
<td></td>
</tr>
<tr>
<td>HX2BIN 03E1</td>
<td>Two ASCII characters on page zero to eight bits.</td>
<td></td>
</tr>
<tr>
<td>BIN2HX 03F2</td>
<td>Four bits to ASCII character on page zero.</td>
<td></td>
</tr>
<tr>
<td>DSPHEX 03FF</td>
<td>Eight bits to two ASCII characters, page zero.</td>
<td></td>
</tr>
<tr>
<td>SYM 040F</td>
<td>Address of symbol table entry X in MISCL, H.</td>
<td></td>
</tr>
<tr>
<td>ADDR 0434</td>
<td>Address for symbol X in ADL, H.</td>
<td></td>
</tr>
<tr>
<td>ADDLAB 0443</td>
<td>Add symbol to table; return number in X.</td>
<td></td>
</tr>
<tr>
<td>NEWSYM 0469</td>
<td>Add symbol if new; call SYM.</td>
<td></td>
</tr>
<tr>
<td>ENCODE 047D</td>
<td>Encode assembly language line; update symbols.</td>
<td></td>
</tr>
<tr>
<td>CMAND 0549</td>
<td>Look up and transfer control to command.</td>
<td></td>
</tr>
<tr>
<td>FIN 0577</td>
<td>Add encoded line to program.</td>
<td></td>
</tr>
<tr>
<td>MAIN 058B</td>
<td>Main program; do command or encode line.</td>
<td></td>
</tr>
<tr>
<td>?BEGIN 0610</td>
<td>Add name to symbols; enter edit mode.</td>
<td></td>
</tr>
<tr>
<td>?ASSGN 062E</td>
<td>Assign addresses to global symbols.</td>
<td></td>
</tr>
<tr>
<td>-LOCAL 0665</td>
<td>Assign addresses to local symbols.</td>
<td></td>
</tr>
<tr>
<td>?REDEFF 0672</td>
<td>Redefine module pointer.</td>
<td></td>
</tr>
<tr>
<td>ASMBL 0683</td>
<td>Translate line into machine code.</td>
<td></td>
</tr>
<tr>
<td>LOCSYM 06CB</td>
<td>Display undefined symbols.</td>
<td></td>
</tr>
<tr>
<td>-ASSEMB 06EB</td>
<td>Assemble module; store at MDL,H.</td>
<td></td>
</tr>
<tr>
<td>?TABLE 071F</td>
<td>Reserve space for arrays.</td>
<td></td>
</tr>
<tr>
<td>INPUT 075D</td>
<td>Prompt with IOBUF; accept input line.</td>
<td></td>
</tr>
<tr>
<td>-STORE 07A6</td>
<td>Save module; clear local symbols; end edit mode.</td>
<td></td>
</tr>
<tr>
<td>*MODLIM 07CC</td>
<td>Lower OPCPTR limit for each address mode.</td>
<td></td>
</tr>
<tr>
<td>DECODE 07D9</td>
<td>Convert source code to ASCII line.</td>
<td></td>
</tr>
<tr>
<td>OUTLIN 08A1</td>
<td>Output line from IOBUF as ASCII.</td>
<td></td>
</tr>
<tr>
<td>PRNTCK 08B1</td>
<td>Check line numbers; print lines if enabled.</td>
<td></td>
</tr>
<tr>
<td>-PRINT 090D</td>
<td>Output lines in range.</td>
<td></td>
</tr>
<tr>
<td>FIXSYM 0917</td>
<td>Revise addresses of symbols in move range.</td>
<td></td>
</tr>
<tr>
<td>INSERT 0940</td>
<td>Open gap in source code for insert; fix symbols.</td>
<td></td>
</tr>
<tr>
<td>-INSRT 0965</td>
<td>Insert and/or delete lines.</td>
<td></td>
</tr>
</tbody>
</table>

* Table.
Table 4.4: Hierarchy of Modules

<table>
<thead>
<tr>
<th>MAIN PROGRAM</th>
<th>?BEGIN</th>
<th>?ASSGN</th>
<th>?REDEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSPHEX</td>
<td>NEWSYM</td>
<td>NEWSYM</td>
<td>LOCSYM</td>
</tr>
<tr>
<td>BIN2HX</td>
<td>MATCH</td>
<td>MATCH</td>
<td>ADDRSS</td>
</tr>
<tr>
<td>INPUT</td>
<td>ADDLAB</td>
<td>ADDLAB</td>
<td>ADDRSS</td>
</tr>
<tr>
<td>ENCODE</td>
<td>SYM</td>
<td>SYM</td>
<td>SYM</td>
</tr>
<tr>
<td>MATCH</td>
<td>HEX</td>
<td>HEX</td>
<td>INPUT</td>
</tr>
<tr>
<td>HX2BIN</td>
<td>ADDRSS</td>
<td>ADDRSS</td>
<td>FIXSYM</td>
</tr>
<tr>
<td>HEX</td>
<td>SYM</td>
<td>SYM</td>
<td>ADDRSS</td>
</tr>
<tr>
<td>ADDLAB</td>
<td>SYM</td>
<td>SYM</td>
<td>SYM</td>
</tr>
<tr>
<td>FIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSERT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIXSYM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRESS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEWSYM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATCH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDLAB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMAND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATCH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRESS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Commands)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?TABLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEWSYM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATCH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDLAB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HX2BIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HEX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-LOCAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-ASSGN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-REDEF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-STORE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-PRINT</td>
<td>PRNTCK</td>
<td>DECODE</td>
<td>FIXSYM</td>
</tr>
<tr>
<td>STORE</td>
<td>SYM</td>
<td>ADDRSS</td>
<td>ADDRSS</td>
</tr>
<tr>
<td>PRINT</td>
<td></td>
<td>SYM</td>
<td>SYM</td>
</tr>
<tr>
<td>PRNTCK</td>
<td></td>
<td>BIN2HX</td>
<td>BIN2HX</td>
</tr>
<tr>
<td>HX2BIN</td>
<td>HEX</td>
<td>DSPHEX</td>
<td>OUTLIN</td>
</tr>
<tr>
<td>HEX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECODE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDRSS</td>
<td>SYM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSPHEX</td>
<td>BIN2HX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIN2HX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUTLIN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5. MODIFICATION

Some users may wish to modify the assembler to expand its capabilities, or for use on another system. Sections 3 and 4 should prove particularly useful to these users. Some comments on specific modifications are given below. To use the assembler on another 650X system, different I/O routines would probably be required. The assembler might also have to be relocated.

5.1 Changing Special Key Definitions

Some terminals lack "escape" or "backspace" keys. Another key may be used by storing its ASCII code at 0776 (2576) for escape, or 0780 (2580) for backspace. Refer to Subroutine INPUT in Section 3.

5.2 Moving Tables

The REGION command temporarily changes the memory location for storage of assembled modules. The assembler can also be permanently modified to store the assembled modules, assembly language, or symbols at a different location.

Initialization value. The location of each array is determined by the initial value of its corresponding pointer. The last line in Table 4.1 gives the source of this initialization value for each array. By changing these values, the array(s) can be initialized to a different location. The current line pointer low order address is always initialized to zero; only the high address can be changed in this way. Both low (first byte) and high (second byte) order addresses can be changed for the other pointers.

Symbol table. The first nine entries (72 bytes) in the symbol table are essential to the assembler, because they are symbols and addresses for the assembler commands. They must be moved if the initialization value for the symbol table is changed. Note that the initialization value points to the ninth symbol, not the first.

5.3 Adding Custom Commands

User commands may be added in the form of subroutines.

Prompt symbols. Command subroutines must return the appropriate prompt symbol in the accumulator: 3F (?) for control mode or 2D (-) for edit mode. Or, an error code may be returned; these must be greater than 3F for control mode, and less than 3F for edit mode. Error codes should be printing ASCII characters.

Adding to symbol table. The ASCII code for the command, beginning with the correct mode prompt symbol, should be entered
in the first six bytes available in the symbol table. This would start at 0A00 (2800) for the first user command. The subroutine address should be stored in the next two bytes, low order first. The initialization value at 02FA, 02FB (20FA,20FB) must be incremented by eight. (See Section 5.2) The initialization value for the top symbol number at 0300 (2100) must be incremented by one.

5.4 Relocation

The assembler may be relocated using a relocation routine such as that in The First Book of KIM. The 0200 version of the assembler starts at address 0200 and ends at 09FF. It contains blocks of data at 0200-03A2, 07CC-07D8, and 09B8-09FF inclusive. The assembler should be relocated an even multiple of 256 bytes, so that it begins at a page boundary, e.g. 0200, 2000, 0400, etc.

The relocation routine mentioned above will correct addresses for subroutine calls, but table references and pointers must be corrected by hand. Since the assembler is relocated an even number of pages, only the high order address must be corrected. For example, to relocate the 0200 version to start at 0800, add six to the number currently at each of the addresses below.

Pointers. Addresses 02ED, 02F4, and 02FB contain initialization values for pointers, as do addresses 02E9 and 02EB.

Command return. The value at address 056B is pushed on the stack as the high order address for return from a command.

Data. Addresses 04A6, 04AE, 04B7, 04BD, 05BD, 068A, 083E, 082F, 0834, 081F, 0848, 084D, and 0852 contain high order addresses for table references.

Symbol table. Each of the first nine entries in the symbol table contains six ASCII characters, corresponding to a command, followed by the low and high order address for the command subroutines. The high addresses, at 09BF to 09FF must be corrected.

5.5 I/O Requirements

The assembler uses standard I/O routines in the KIM monitor. Functionally equivalent user routines may be substituted for use with another I/O device or 6502 system. Table 5.1 gives a brief description of each of these routines, together with the addresses of lines in the assembler which call each subroutine.
Table 5.1: I/O Routines

<table>
<thead>
<tr>
<th>KIM Routine</th>
<th>Function</th>
<th>Assembler References</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRLF</td>
<td>Carriage return, line feed</td>
<td>075D (255D)</td>
</tr>
<tr>
<td>1E2F</td>
<td></td>
<td>08A1 (26A1)</td>
</tr>
<tr>
<td>OUTCH</td>
<td>Output ASCII from A.</td>
<td>0764 (2564)</td>
</tr>
<tr>
<td>1EAO</td>
<td>Preserve X.</td>
<td>08A8 (26A8)</td>
</tr>
<tr>
<td>GETCH</td>
<td>Input ASCII to A.</td>
<td>0772 (2572)</td>
</tr>
<tr>
<td>1E5A</td>
<td>Preserve X.</td>
<td></td>
</tr>
<tr>
<td>OUTSP</td>
<td>Output one space.</td>
<td>078D (258D)</td>
</tr>
</tbody>
</table>

Figure A.1: Keyboard Interface
APPENDIX A: AN INEXPENSIVE I/O SYSTEM

Many 6502 users, myself included, do not have a computer terminal. I have developed a very inexpensive "terminal substitute." I use a $30 unencoded keyboard for input, and display a 64-character ASCII subset on the KIM-1 display.

The keyboard is scanned using software, which allows keys and combinations of keys to be defined arbitrarily. For example, multiple key depressions could be used for playing chords in music synthesis applications. The I/O software given here simulates a simple ASCII keyboard with "shift" but without "control" or "repeat." The required software decreases the space available for program storage. Using the KIM-1 display for output of ASCII characters can be frustrating, but it is a big improvement over no ASCII output at all. The keyboard interface might also be of interest to those planning to add one of Lancaster's "cheap video" displays.

Keyboard interface. Figure A.1 is a schematic for the keyboard interface. The unencoded keyboard must be wired as a matrix of eight rows and eight columns. One CMOS 4051 is used as a multiplexer and the other as a demultiplexer. Output lines PA0 to PA5 select the row and column of interest. PA7 goes low if the corresponding key is depressed.

The "shift" key must be connected to channel 7 of each 4051. Channel 0 of one 4051 must be shorted to channel 0 of the other. Other row and column assignments are arbitrary, since assignment of ASCII codes is done in software.

The keyboard, 4051 chips, and wire-wrap sockets are available from Jameco Electronics, 1021 Howard Ave., San Carlos, CA 94070 for under $35. They also sell a wire-wrapping kit for $13.

Testing the interface. Load and run the relocatable test routine below. With no key depressed, the data display should read 00. Pressing the "shift" key should cause 3F to be displayed. If not, the keyboard interface is connected incorrectly. When another key is pressed, the hexadecimal code for its row and column will be displayed. Record this key number for each key. Then make a table giving the ASCII equivalent for each key number from 00 to 3F. Key numbers 00 and 3F correspond to "end of scan" and "shift," respectively, so the value entered for them will be ignored. This 64 byte table should be loaded at address 0E80. There may be more than one key for a given ASCII code, and not all ASCII codes will be used.

I/O routines. Next, load the rest of the I/O software, beginning with Table SECTAB and ending with Subroutine CRJF. SECTAB gives the pattern of lit segments to display a 64 character ASCII subset (ASCII 20 through 5F) on the KIM-1 display.
Some characters will look strange at first, but recognition becomes easy with very little practice. The subroutines GETCH, OUTCH, OUTSP, and CRLF are functionally equivalent to the KIM monitor routines of the same names. Their addresses must be substituted in the assembler I/O subroutine calls as explained in Section 5.5. These routines could also be used in other terminal-based programs.

Listing A. Test program for Qwerty keyboard. Displays hexadecimal code of active key.

A9 7F          LDA#   7F          Define I/O.
8D 01 17       STA     PADD
A9 00          LDA#   00          Initialize pointer
85 FA          STAZ    POINTL    for display routine.
A9 17          LDA#   17
85 FB          STAZ    POINTH
A9 40          START   LDA#   40    Scan 63 keys.
8D 00 17       STA     PAD
CE 00 17       SCANKB  DEC    PAD    Find active key.
AD 00 17       LDA     PAD
30 F8          BMI     SCANKB
20 19 1F       JSR     SCAND    Display key.
18              CLC
90 ED          BCC     START    Repeat for new key.
EA              NOP
Table SEGTAB. Seven-segment code to display 64-character ASCII subset. Modify as desired.

00 0A 22 1B 36 24 5F 02 39 0F 21 18 0C 40 08 52
3F 06 5B 4F 66 6D 07 7F 6F 41 45 60 48 42 53
7B 77 7C 58 5E 79 71 3D 76 04 1E 70 38 37 54 5C
73 67 50 2D 78 1C 6A 3E 14 6E 49 39 44 0F 77 61

Subroutine DSPLY. Display 6 characters on KIM readout for about 3 msec.

A9 7F LDA# 7F Define I/O.
8D 41 17 STA PCDD
A9 15 LDA# 15 Initialize char.
8D 42 17 STA PDD
A2 05 LDX# 05 Display 6 chars.
CE 42 17 CHAR DEC PDD Select next char.
CE 42 17 DEC PDD
B5 23 LDAZX DSPBUF Get segment code.
8D 40 17 STA PCD Turn segments on.
A0 64 LDY# 64 Wait 500 msec.
88 WAIT DEY
10 FD BPL WAIT
A9 00 LDA# 00 Turn segments off.
8D 40 17 STA PCD
CA DEX
10 E8 BPL CHAR Another char?
60 RTS
0F25 Subroutine GETKEY. Scan kybd; return ASCII in A, key in Y.

A2 3F LDX# 3F Define I/O.
8E 01 17 STX PADD
8E 00 17 STX PAD
CE 00 17 NXTKEY DEC PAD Scan 2 keys.
AD 00 17 LDA PAD for active key.
30 F8 BMI NXTKEY
29 3F AND# 3F Mask input bit.
A8 TAY Return if no key.
D0 01 BNE ANYKEY
60 RTS
B9 80 0E ANYKEY LDAY KEYTAB Get ASCII.
8E 00 17 STX PAD Check shift key.
2C 00 17 BIT PAD
10 01 BPL SHFTKY
60 RTS No shift; return.
C9 21 SHFTKY CMP# 21 shift legal?
10 01 BPL NOT2LO
60 RTS
C9 40 NOT2LO CMP# 40
30 01 BMI NOT2HI
60 RTS
49 10 NOT2HI EOR# 10 Find shift char.
60 RTS
0F54 Subroutine ADDCH. Shift ASCII character in A into display from right.

A2 00 LDX# 00     Shift display
B4 24  LEFT LDYX   DSPBFI
94 23  STYZX DSPBUF
E8 INX
E0 05 CPX3 05
30 F7 BMI LEFT
E9 20 SBC# 20     Find segment
AA TAX SEGTAB
BD C0 0E LDAX DSPBF5
85 28 STAZ Add at right.
60 RTS

0F68 Subroutine GETCH. Get character from keyboard. Return ASCII in A. Add to display or backspace as required. X is preserved.

B6 3B STXZ SAVX     Save X.
20 00 0F OLD JSR DSPLAY Wait for release
20 25 OF JSR GETKEY of old key.
D0 F8 BNE OLD
EA NOP
20 00 0F NONE JSR DSPLAY Wait for new
20 25 OF JSR GETKEY key depressed.
F0 F8 BEQ NONE
C9 08 CMP# 08      Backspace?
D0 10 BNE NOTBSP
A2 04 LDX# 04      Yes. Shift
B4 23  RIGHT LDYX DSPBUF display right.
94 24 STYZX DSPBFI
CA DEX
10 F9 BPL  RIGHT
A0 00 LDY# 00     Add blank
84 23 STYZ DSPBUF at left.
A6 3B LDXZ SAVX   Restore X.
60 RTS

48 NOTBSP PHA Else, add char
20 54 0F JSR ADDCH to display.
A6 3B LDXZ SAVX
68 PLA
60 RTS
OF97 Subroutine OUTCH. Add ASCII character in A to display. Display for about 0.2 sec. Preserve X.

86 3B          STXZ          SAVX          Save X.
20 54 0F        JSR ADDCH    Add char.
A9 40          LDA# 40      Wait 0.2 sec
85 5C          STAZ TIME    before returning.
20 00 0F SHOW   JSR DSPLAY
C6 5C          DECZ TIME
10 F9          BPL SHOW
A6 3B          LDXZ SAVX    Restore X.
60             RTS

OFAA Subroutine OUTSP. Output one space.

A9 20          LDA# 20
20 97 0F        JSR OUTCH
60             RTS

OFB0 Subroutine CRLF. Clear display.

A9 00          LDA# 00
A2 05          LDX# 05
95 23 CLEAR    STAZX DSPBUF
CA             DEX
10 FB BPL CLEAR
60             RTS
APPENDIX B: ANSWERS TO USER QUESTIONS

Q. Can the assembler be stored in read only memory?

A. Yes; it will just fit in a 2K ROM. Presumably it will have to be relocated, following the instructions in Section 5.4. In addition, the assembler must be entered at the relocated equivalent of 09AA. This routine, which is unused in the RAM version of the assembler, transfers the first nine entries in the symbol table from ROM to RAM. These symbols correspond to commands and are essential to the assembler. The correct source and destination addresses must be substituted in this initialization routine. Permission to reproduce the assembler in ROM may be obtained from the author.

Q. If I have enough memory, can I expand the symbol table?

A. Yes. The standard version of the assembler allows 64 symbols, including nine for assembler commands. Space is available for nine additional symbols if overflow error detection is defeated by setting 05B4 (23B4) = EA. The assembler can also be modified to give an overflow error message when the number of symbols exceeds 128, by setting 05B0 (23B0) = 10. Expanding the symbol table to 128 entries requires moving the module and assembled program storage areas. See Section 5.2. Actually, quite lengthy programs can be assembled within the limit of 55 user symbols, since local symbols are cleared each time a module is stored.

Q. My video terminal only has 32 characters per line, so your print routine runs over by one character. Any advice?

A. Make the following changes at the addresses indicated: 0870 (2670) = 14, 0880 (2680) = 14, 0890 (2690) = 1B, 08AD (26AD) = 20, 08ED = 1B. Input lines may still exceed 32 characters.

Q. Can the assembler be used with the SYM microcomputer?

A. Easily. The I/O routine addresses must be changed as explained in Section 5.5. The SYM monitor addresses are 834D (CRLF), 8A47 (OUTCH), 8A1B (GETCH), and 8342 (OUTSP).

Q. How about a command to give the starting address of the module without having to check 0040, 0041?

A. This is just one example of a number of commands that could easily be implemented by users who don't insist on fitting the assembler in a 2K ROM. It is also possible to add features by sacrificing existing commands. For example, some users may rarely use ?REDEF. Others may use ?ASSGN and ?REDEF to name and reserve space for tables. Either command could be replaced by a user-written command. Reviewers disagreed on some of the most desired features in a 2K assembler. The assembler is sufficiently easy to modify that the final choice can be left to the user.
A541   --TEST   LDAZ   MLDADH  00
A202   LDX#   02   02
20FF03  JSR   DSPHEX  04
A540   LDAZ   MLDADL  07
A204   LDX#   04   09
20FF03  JSR   DSPHEX  0B
20A100  JSR   OUTLIN  0E
A21A   LDX#   1A   11
B53C   SAVE   LDAZX   GLOBAL  13
9DE00B  STAX   COPY   15
CA     DEX   18
10F8   BPL   SAVE   19
4C001C  JMP   MONITR  1B
A21A   ENTER   LDX#   1A   1C
BDE00B  RESTR  LDX   COPY   20
953C   STAZX   GLOBAL  23
CA     DEX   25
10F8   BPL   RESTR  26
4CDS05  JMP   WARM  28
2K SYMBOLIC ASSEMBLER: REVISIONS

Here are the corrections for all bugs found so far, along with some optional modifications to the 2KSA.

BACKSPACE BUG

The "backspace" key does not delete the last character, but only moves a pointer to allow typing over it. It is not possible to blank out a character using the "space" key, because that is used to advance it to the next field. One solution is to use "tab" to advance to the next field, freeing "space" for use as a blanking character. (Thanks to Nelson Edwards for finding this bug.)

ADDRESS ASSIGNMENT PROBLEMS

The 2KSA is designed to prevent accidental re-assignment of an address to a symbol. Early versions were a bit overzealous in this area, and should be fixed by loading at 0478: 34, 04, C9, FF. The re-assignment check can also be defeated completely, if desired, by loading at 047A: A9, 00. Just don't forget and use the same symbol twice.

EASIER RELOCATION

Relocation of modules in edit mode is possible if ?REDEF is changed to -REDEF. Set 09D0=2D and 0661=2D.

EASIER TESTING

The command --TEST (facing page) can be used to print the start address of the module and leave the assembler for testing. The extra hyphen is required because the I/O buffer isn't cleared. --TEST also automatically saves the pointers required for source code storage starting at address 0BE0. Source code can then be saved by simply dumping 0A00-0CB0.

The listing also contains a re-entry routine (starting at ENTER) which restores the pointers before entering edit mode. This would ordinarily be used after loading source code from tape.

To substitute --TEST for ?TABLE, load it at 071F and load at 05E0: 2D, 2D, 54, 45, 53, 54. MONITR should be the warm start address for the monitor of your particular computer.
SOURCE CODE TAPE RECORD FORM

To save:

Record pointer values below.
Dump OA00 through OC7F.

To retrieve:

Initialize assembler.
Hit reset.
Load module from tape.
Restore pointers.
Enter assembler from 05D6.
Ignore any error code.

<table>
<thead>
<tr>
<th>Module Name</th>
<th>ID</th>
<th>GLOBAL</th>
<th>PRGLEN</th>
<th>SYMTBL</th>
<th>SYMNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>003C</td>
<td>003D</td>
<td>0050,51</td>
<td>0056</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Permission is hereby granted to photocopy this page.
x"Jd<"留W"
1>7"I"JL
0:"Jt%:=(*
1>8&)/*
-ee**=
= 毛7I"f/
1w"%7I*1"
1>*/
1W"17p"
"M"5Gh"0u
")%QWXA"
a Y"aZ)9
y%>EYPF9E2PF9
EYO EZ $80'
jj%We/Uf
00 01 02 03 04 05 06 07 08 09 OA OB OC OD OE OF
0800 F8 A2 01 CA E4 3C 10 E0 A0 00 B1 57 A2 00 FF D5
0810 03 B1 3E 85 37 A2 0C CA C9 1D 10 02 A2 01 DD CC 07,7C
0820 30 04 86 3A A2 00 CA 10 FA 53 0A AA BB AD A8 02,DA
0830 85 11 BD A9 02 85 12 B1 3E 38 A6 3A FD DC 02 85,06
0840 2D 0A 18 65 2D AA BD 00 02 85 OE BD 01 02 85 0F,07
0850 BD A9 02 85 10 A5 37 C9 1D 10 01 60 E6 2F AO 01,46
0860 B1 57 A2 02 20 FF 03 A5 37 C9 2A 10 08 B1 3E A2,0C
0870 15 20 FF 03 B1 3E AA 20 FF 04 A0 05 B1 2B 99,09
0880 15 00 88 10 FF A5 37 C9 69 10 01 60 E6 2F AO 02,E4
0890 B1 A2 02 40 FF 03 B1 3E FF 05 A2 1C 20 FF FF 03,7B
08A0 60 20 4D 83 00 B5 00 20 47 8A EE E0 23 30 F6,21
08B0 A9 09 00 85 3E A5 40 85 57 A5 41 85 58 A2 07 20,3A
08C0 E1 03 86 59 A2 02 B0 20 E1 03 86 5A A9 02 85 39 20,17
08D0 D9 07 A5 3E C5 59 DD 02 C6 39 C5 5A D0 02 C6 39,B9
08E0 C5 59 30 12 C5 5A 10 OD 24 38 30 08 A2 1F 20 FF,C9
08F0 C5 20 A1 08 EA 18 A5 57 65 2F 85 57 90 02 E6,65
00 01 02 03 04 05 06 07 08 09 OA OB OC OD OE OF
0900 58 18 A5 3E 65 2F 85 3E C5 5D 30 C3 60 A9 01 85,93
0910 38 20 B1 08 A9 2D 60 A6 56 20 34 04 C5 3F D0 1A,1C
0920 A5 3E C5 3D 30 13 A4 29 C4 5A 10 06 A9 FE A0 07,7F
0930 91 2B 18 65 2F A0 06 91 2B EA CA E4 3C 10 DA 60,67
0940 2F 85 29 A5 3F 85 2A A5 3D 38 E5 3E A8 B1 3E 91,D1
0950 29 88 10 F9 60 A9 FF 85 38 20 B1 08 C5 5A D0 02,1A
0960 2F 85 29 A5 3F 85 2A A5 3D 38 E5 3E A8 B1 3E 91,D1
0970 C6 39 A5 39 FF 03 A9 25 60 A5 59 85 3E A6 5A FO,C9
0980 26 38 E5 5A 85 2F 20 17 09 A5 3F 85 58 A5 3D 38,3B
0990 E5 3E 85 2D A5 3D 18 65 2F 85 3D A0 00 B1 5A 91,99
a1 <== e=a=E12
09B0 B8 09 CA 10 F7 4C B8 05 3F 41 53 53 47 4E 2E 06,65
09C0 3F 42 45 47 49 4E 10 06 2D 4C 4F 43 41 4C 65 06,22
09D0 3F 52 45 44 45 46 72 06 2D 4C 4F 43 41 4C EB 06,06
09E0 3F 54 41 42 4C 45 1F 07 2D 53 54 4F 52 45 A6 07,0A
09F0 2D 50 52 49 4E 54 0D 09 2D 49 4E 53 52 54 65 09,05
X%>E/e/>E=0C/}
8 1->~&V 4 E?P
%E0 $D2 )~
+E/ +jJd<"Z' 
/E=P` Z%e
/ %A *%=Be>(1) 
Y' >8 1 E2P
F9%P)%%Y&Zp
&8e2 / %? [%=8 
->-%= e= 12
A0 3E C8 C4 2D 30 F7 EA A9 2D 60 A2 47 BD BB 09 9D, DB
HD-0wj)-"G=8
8 J wL8 ?ASSGN. 
?BEGIN -LOCALe
?REDEFr -ASSEM&
?TABLE -STORE&
-PRINT -INSRTe