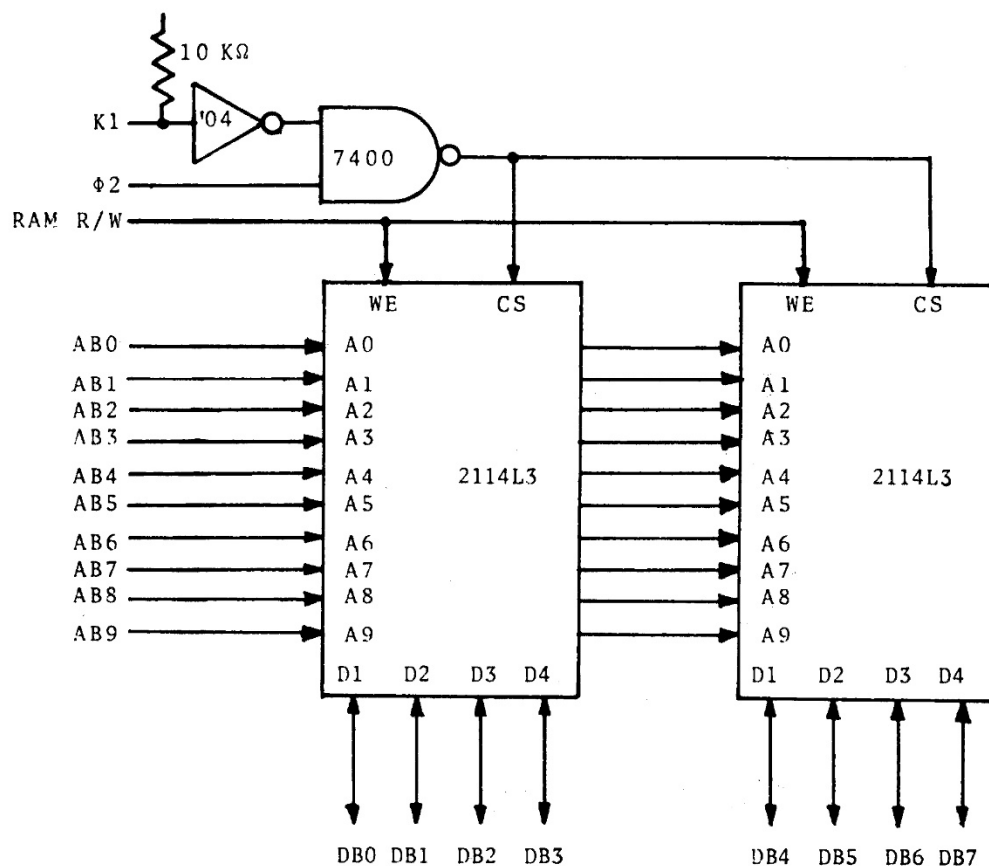


A 2K SYMBOLIC ASSEMBLER FOR THE 6502



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 0C80 (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, indexed 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 2.1: Input Format for Commands and Instructions

	Field 1	Field 2	Field 3	Field 4	Field 5
Assign address to symbol.	?ASSGN	symbol	nnnn		
Reserve space for table.	?TABLE	symbol	nn		
Redefine module pointer.	?REDEF	nnnn			
Begin new module.	?BEGIN	symbol			
One-byte instructions.		(symbol)	opcode		
Immediate mode instructions.		(symbol)	opcode	nn	
Other two-byte instructions.		(symbol)	opcode	symbol	
Three-byte instructions.		(symbol)	opcode	symbol	(nn)
Define local symbol.	-LOCAL	symbol	nnnn		
Assemble module.	-ASSEM				
Print lines in range.	-PRINT	nnT0nn			
Insert before line given.	-INSRT	nn			
Replace lines in range.	-INSRT	nnT0nn			
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 overwrite 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: 003C, 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.
- 4 Operand undefined; must be on page zero.
- 5 Operand not on page zero.
- 6 Offset legal for three-byte instructions only.
- 7 Relative branch illegal outside module.
- 8 Absolute addressing illegal within module.
- 9 Command 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
- OC00      LDX#      2F
1 OC00      LDX#      2F
- OC02      DEX
- OC03      BPL      LOOP
- OC05      RTS
- OC06 -ASSEM
LOOP
- OC06 -INSRT  02T003
- OC02      LOOP      DEX
- OC03 -ASSEM
- OC06 -PRINT  00T006
A22F DELAY  LDX#      2F      00
CA    LOOP  DEX      02
10FD   BPL      LOOP      03
60     RTS      05
- OC06 -STORE
?      ?REDEF  0070
?      ?BEGIN  WAVGEN
- OC00 -LOCAL  BASE    0061
-LOCAL
- OC00      LDYZ      PERIOD
- OC02      LOOP      LDAY      WAVE
- OC05      ADCIY     BASE
- OC07      STA       PAD      02
- OC0A      JSR       DELAY
- OC0D      DEY
- OC0E      BNE      LOOP
- OC10      RTS
- OC11 -ASSEM
- OC11 -PRINT  00T011
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
DOF2      BNE      LOOP      0E
60        RTS      10
- OC11 -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	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	39	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
A2 00		LDX#	00	search parameter
86 2A		STXZ	ADH	list in ADL, H.
A0 06		LDY#	06	
B1 29	PARAM	LDAIY	ADL	Move parameters
99 30 00		STAY	TBL	to workspace.
88		DEY		
10 F8		BPL	PARAM	
A6 36		LDXZ	NUM	Compare X records.
A4 35	RECORD	LDYZ	HBC	
B1 30	BYTE	LDAIY	TBL	First Y+1 bytes
D1 32		CMPIY	RFL	must match.
F0 02		BEQ	OK	
A0 FF		LDY#	FF	Mismatch.
88	OK	DEY		
10 F5		BPL	BYTE	
C8		INY		All ok?
D0 01		BNE	INCADR	
60		RTS		z set.
38	INCADR	SEC		
A5 30		LDAZ	TBL	Find base address
E5 34		SBCZ	LEN	of next record.
85 30		STAZ	TBL	
B0 02		BCS	DECNUM	
C6 31		DECZ	TBH	
CA	DECNUM	DEX		
10 E1		BPL	RECORD	Last record?
60		RTS		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	
38		SEC		Letter; adjust.
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	
E8		INX		and low byte.
20 D5 03		JSR	HEX	
05 2D		ORAZ	TEMP	Combine.
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	
18		CLC		Letter; adjust.
69 07		ADC#	07	
18	NUMER	CLC		Convert to ASCII.
69 30		ADC#	30	
95 00		STAZX	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	
E8		INX		Find low character.
68		PLA		
29 0F		AND#	0F	
20 F2 03		JSR	BIN2HEX	
60		RTS		

040F Subroutine SYM. Puts base address of symbol table entry X in MISCL, H.

38		SEC		Find difference
86 2D		STXZ	TEMP	between last
A5 56		LDAZ	SYMNUM	record and X.
E5 2D		SBCZ	TEMP	
85 2B		STAZ	MISCL	
A9 00		LDA#	00	
85 2C		STAZ	MISCH	
18		CLC		
A0 02		LDY#	02	
26 2B	X8	ROLZ	MISCL	Multiply by 8
26 2C		ROLZ	MISCH	bytes per record.
88		DEY		
10 F9		BPL	X8	
38		SEC		Subtract from
A5 50		LDAZ	SYMTBL	address of
E5 2B		SBCZ	MISCL	last record.
85 2B		STAZ	MISCL	
A5 51		LDAZ	SYMTBH	
E5 2C		SBCZ	MISCH	
85 2C		STAZ	MISCH	
60		RTS		

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

20 0F 04		JSR	SYM	Get base address.
A0 06		LDY#	06	Get symbol address.
B1 2B		LDAIY	MISCL	
85 29		STAZ	ADL	Put in ADL, H.
C8		INY		
B1 2B		LDAIY	MISCL	
85 2A		STAZ	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	ADL,H points
A9 00		LDA#	00	to symbol.
85 2A		STAZ	ADH	
18		CLC		
A5 50		LDAZ	SYMTBL	Find new base
69 08		ADC#	08	address of
85 50		STAZ	SYMTBL	symbol table.
90 02		BCC	NOADDR	
E6 51		INCZ	SYMTBH	
A0 07	NOADDR	LDY#	07	
A9 FF		LDA#	FF	Set high address
91 50		STAIY	SYMTBL	=FF (unassigned).
88		DEY		
88		DEY		
B1 29	XFRSYM	LDAIY	ADL	Add symbol to
91 50		STAIY	SYMTBL	symbol table.
88		DEY		
10 F9		BPL	XFRSYM	
A6 56		LDXZ	SYMNUM	Increment number
E8		INX		of symbols.
86 56		STXZ	SYMNUM	
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	Set up search.
A2 50		LDX#	50	
20 A3 03		JSR	MATCH	Look up symbol.
F0 05		BEQ	OLD	
A5 52		LDAZ	SYMRFL	Not found; add
20 43 04		JSR	ADDLAB	to symbol table.
20 0F 04	OLD	JSR	SYM	Address in MISCL, H.
E4 56		CPXZ	SYMNUM	Set z if new.
60		RTS		

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	
20 A3 03		JSR	MATCH	Find address mode.
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	
10 02		BPL	NOTIMP	
A2 00		LDX#	00	Implied mode.
C9 30	NOTIMP	CMP#	30	
30 02		BMI	NOTREL	
A2 08		LDX#	08	Relative mode.
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	
A9 33		LDA#	33	"3" Too low.
60		RTS		
DD CF 02	NOT2LO	CMPX	MAX	
30 03		BMI	NOT2HI	
A9 33		LDA#	33	"3" Too high.
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		Continue.

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	Consider opcode.
C9 1D		CMP#	1D	
10 03		BPL	OPRRQD	Operand required?
A9 2D		LDA#	2D	"_"
60		RTS		No; return.
E6 2F	OPRRQD	INCZ	BYTES	At least 2 bytes.
C9 2A		CMP#	2A	
10 0A		BPL	NOTIMM	
A2 15		LDX#	15	Immediate addressing.
20 E1 03		JSR	HX2BIN	Find binary value
86 38		STXZ	SYMPTR	
A9 2D		LDA#	2D	"_"
60		RTS		
A2 15	NOTIMM	LDX#	15	Set up operand search.
86 52		STXZ	SYMRFL	
C9 61		CMP#	61	
10 20		BPL	NOTZPG	Zpage addressing?
A2 50		LDX#	50	Yes.
20 A3 03		JSR	MATCH	Look up operand.
F0 03		BEQ	FOUND	
A9 34		LDA#	34	"4" Not found.
60		RTS		
20 34 04	FOUND	JSR	ADDRSS	
F0 03		BEQ	OK	
A9 35		LDA#	35	"5" Not zpage.
60		RTS		
86 38	OK	STXZ	SYMPTR	Store operand.
A5 1C		LDAZ	OFFSET	Check for offset.
C9 20		CMP#	20	"SP"
F0 03		BEQ	DONE	
A9 36		LDA#	36	"6" offset illegal.
60		RTS		
A9 2D	DONE	LDA#	2D	"_"
60		RTS		OK, return.
EA	NOTZPG	NOP		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	
C9 20		CMP#	20	"SP"
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	"_" Stay in
60		RTS		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	ADDRSS	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	LDAY	OPCPTR	Add line
91 3E		STAIY	CRNTAL	to program.
88		DEY		
10 F8		BPL	ADDLIN	
A5 07		LDAZ	LABEL	
C9 20		CMP#	20	"SP"
F0 10		BEQ	INCADR	Any label?
A9 07		LDA#	07	Yes. Add to
20 69 04		JSR	NEWSYM	symbol table
A0 07		LDY#	07	if new, and
A5 3F		LDAZ	CRNTAH	assign address.
91 2B		STAIY	MISCL	
88		DEY		
A5 3E		LDAZ	CRNTAL	
91 2B		STAIY	MISCL	
18	INCADR	CLC		
A5 3E		LDAZ	CRNTAL	Increment pointers.
65 2F		ADCZ	BYTES	
85 3E		STAZ	CRNTAL	
18		CLC		
A5 3D		LDAZ	PRGLEN	
65 2F		ADCZ	BYTES	
85 3D		STAZ	PRGLEN	
10 03		BPL	OK	
A9 42		LDA#	42	"B" Error-
60		RTS		program overflow.
24 56	OK	BITZ	SYMNUM	
50 03		BVC	OK2	
A9 43		LDA#	43	"C" Error-
60		RTS		symbol overflow.
A9 2D	OK2	LDA#	2D	
60		RTS		

05B8 Main program. Process command, or translate input into source code.

D8		CLD		
A2 18		LDX#	18	Initialize
BD E9 02	INIT	LDAZ	PRMTAB	program parameters.
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	IOBUF1	Clear I/O buffer
CA		DEX		except error code.
10 FB		BPL	CLEAR	
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	IOBUF1	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		

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

A9 07		LDA#	07	Add name to
20 69 04		JSR	NEWSYM	symbol table.
F0 03		BEQ	OK	
A9 44		LDA#	44	"D" Error-
60		RTS		label in use.
86 3C	OK	STXZ	GLOBAL	Set local cutoff.
A9 00		LDA#	00	Clear pointers.
85 3E		STAZ	CRNTAL	
85 3D		STAZ	PRGLEN	
A0 06		LDY#	06	
91 2B		STAIY	MISCL	Set start address
A5 3F		LDAZ	CRNTAH	=CRNTAL, H.
C8		INY		
91 2B		STAIY	MISCL	
A9 2D		LDA#	2D	"-" Set
60		RTS		input mode.

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	
20 69 04		JSR	NEWSYM	Add symbol to table.
F0 03		BEQ	NOTOLD	
A9 44		LDA#	44	"D" Error-
60		RTS		label in use.
A2 0E	NOTOLD	LDX#	0E	Assign address.
20 E1 03		JSR	HX2BIN	
A0 07		LDY#	07	
8A		TXA		
91 2B		STAIY	MISCL	
A2 10		LDX#	10	
20 E1 03		JSR	HX2BIN	
88		DEY		
8A		TXA		
91 2B		STAIY	MISCL	
A9 20		LDA#	20	"SP"
A2 0C		LDX#	0C	clear I/O buffer
95 07	CLEAR	STAZX	LABEL	except prompt.
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
C9 44	CMP#	44	symbol table
D0 03	BNE	OK	if new.
A9 3A	LDA#	3A	":" Error-
60	RTS		symbol in use.
A9 2D	OK	LDA# 2D	"-" stay in
60	RTS		input mode.

0672 ?REDEF. Redefine module start address.

A2 07	LDX#	07	Find high address.
20 E1 03	JSR	HX2BIN	
86 41	STXZ	MDLADH	Store.
A2 09	LDX#	09	Find low address.
20 E1 03	JSR	HX2BIN	
86 40	STXZ	MDLADL	Store.
A9 3F	LDA#	3F	"?" stay in
60	RTS		command mode.

0683 Subroutine ASMBL. Translate line into machine code;
store result at (OBJECT). Return length-1 in Y.

A0 00		LDA#	00	Get first byte.
B1 3E		LDAIY	CRNTAL	
AA		TAX		
BD 05 03		LDAX	OPCTAB	Look up opcode.
91 57		STAIY	OBJECT	
E0 1D		CPX#	1D	
10 01		BPL	OPREQ	
60		RTS		No operand.
C8	OPREQ	INY		
B1 3E		LDAIY	CRNTAL	
E0 2A		CPX#	2A	
10 03		BPL	NOTIMM	Address mode?
91 57		STAIY	OBJECT	Immediate.
60		RTS		
86 2E	NOTIMM	STXZ	MNE	
AA		TAX		
20 34 04		JSR	ADDRSS	Get address.
A5 29		LDAZ	ADL	
A0 01		LDY#	01	
A6 2E		LDXZ	MNE	
E0 61		CPX#	61	
10 03		BPL	NOTZPG	
91 57		STAIY	OBJECT	Zero page.
60		RTS		
E0 69	NOTZPG	CPX#	69	
10 09		BPL	NOTREL	
38		SEC		Relative.
E9 02		SBC#	02	Compute branch.
38		SEC		
E5 3E		SBCZ	CRNTAL	
91 57		STAIY	OBJECT	
60		RTS		
18	NOTREL	CLC		Absolute.
C8		INY		
71 3E		ADCIY	CRNTAL	Add offset.
88		DEY		
91 57		STAIY	OBJECT	
C8		INY		
A5 2A		LDAZ	ADH	
69 00		ADC#	00	
91 57		STAIY	OBJECT	
60		RTS		

06CB Subroutine LOCSYM. Displays undefined local symbols.

A6 3C		LDXZ	GLOBAL	For local symbols,
E8	NXTSYM	INX		
20 34 04		JSR	ADDRSS	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 (MDLADL, H).

20 CB 06		JSR	LOCSYM	Check for local
A9 2D		LDA#	2D	undefined symbols.
C5 00		CMPZ	IOBUF	
F0 01		BEQ	ALLOK	If any; return.
60		RTS		
A9 00	ALLOK	LDA#	00	Else, assemble.
85 3E		STAZ	CRNTAL	Initialize pointers.
A5 40		LDAZ	MDLADL	
85 57		STAZ	OBJECT	
A5 41		LDAZ	MDLADH	
85 58		STAZ	OBJCT1	
20 83 06	NEXTLN	JSR	ASMBL	Translate a line.
84 2D		STYZ	TEMP	Save bytes -1.
38		SEC		Increment pointers.
A5 57		LDAZ	OBJECT	For object code.
65 2D		ADCZ	TEMP	
85 57		STAZ	OBJECT	
90 02		BCC	SKIP	
E6 58		INCZ	OBJCT1	
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		CLC		
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	<u>2F</u>	<u>1E</u>		JSR	CRLF		New line.
A2	00			LDX#	00		Prompt w/
B5	00		PROMPT	LDAZX	IOBUF		first 6 chars.
20	<u>A0</u>	<u>1E</u>		JSR	OUTCH		
E8				INX			
E0	06			CPX#	06		
30	F6			BMI	PROMPT		
A2	00			LDX#	00		Initialize pointer.
A9	06			LDA#	06		7 chars/word
85	2D			STAZ	TEMP		includes space.
20	<u>5A</u>	<u>1E</u>	START	JSR	GETCH		Input a char.
C9	<u>1B</u>			CMP#	1B		"ESC"
D0	01			BNE	NOTBRK		
00				BRK			Break.
C9	<u>0D</u>		NOTBRK	CMP#	0D		"CR"
D0	01			BNE	NOTCR		
60				RTS			End of line.
C9	<u>08</u>		NOTCR	CMP#	08		"BS"
D0	05			BNE	NOTBSP		
CA				DEX			Backspace.
E6	2D			INCZ	TEMP		
A9	08			LDA#	08		
C9	<u>20</u>		NOTBSP	CMP#	20		"SP"
D0	0D			BNE	NOTSP		
EA				NOP			Next word.
20	<u>9E</u>	<u>1E</u>	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 symbols from symbol table.
20 0F 04		JSR	SYM	
86 56		STXZ	SYMNUM	
A5 2B		LDAZ	MISCL	
85 50		STAZ	SYMTBL	
A5 2C		LDAZ	MISCH	
85 51		STAZ	SYMTBH	
A0 07		LDY#	07	Assign address to module.
A5 41		LDAZ	MDLADH	
91 2B		STAIY	MISCL	
88		DEY		
A5 40		LDAZ	MDLADL	
91 2B		STAIY	MISCL	
18		CLC		
65 3D		ADCZ	PRGLEN	Increment MDLADL,H by length of module.
85 40		STAZ	MDLADL	
90 02		BCC	SKIP	
E6 41		INCZ	MDLADH	
A9 3F	SKIP	LDA#	3F	"?" Return to command mode.
60		RTS		

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

0815 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	
A5 3A		LDAZ	MODE	Put mode in
0A		ASLA		I/O buffer.
AA		TAX		
BD A8 02		LDAX	MODTAB	
85 11		STAZ	OPCOD3	
BD A9 02		LDAX	MODTAB 01	
85 12		STAZ	OPCOD4	
B1 3E		LDAIY	CRNTAL	Find mnemonic.
38		SEC		
A6 3A		LDXZ	MODE	
FD DC 02		SBCX	BASE	Mnemonic number.
85 2D		STAZ	TEMP	Multiply by 3.
0A		ASLA		
18		CLC		
65 2D		ADCZ	TEMP	
AA		TAX		Get ASCII.
BD 00 02		LDAX	MNETAB	Put mnemonic in
85 0E		STAZ	OPCODE	I/O buffer.
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	BYTES	At least 2 bytes.

085E Subroutine DECODE (part 3). Decode operands and offset, if any.

A0 01		LDY#	01	
B1 57		LDAIY	OBJECT	Machine code
A2 02		LDX#	02	for operand in
20 FF 03		JSR	DSPHEX	I/O buffer.
A5 37		LDAZ	OPCPTR	
C9 2A		CMP#	2A	Immediate mode?
10 08		BPL	NOTIMM	
B1 3E		LDAIY	CRNTAL	Yes; put hex
A2 15		LDX#	15	number in
20 FF 03		JSR	DSPHEX	I/O buffer.
60		RTS		
B1 3E	NOTIMM	LDAIY	CRNTAL	No; look up
AA		TAX		operand.
20 0F 04		JSR	SYM	
A0 05		LDY#	05	Put operand
B1 2B	SHOWOP	LDAIY	MISCL	in IOBUF.
99 15 00		STAY	OPRAND	
88		DEY		
10 F8		BPL	SHOWOP	
A5 37		LDAZ	OPCPTR	3-byte instruction.
C9 69		CMP#	69	
10 01		BPL	ABS	
60		RTS		No; done
E6 2F	ABS	INCZ	BYTES	Yes.
A0 02		LDY#	02	
B1 57		LDAIY	OBJECT	Add code to
A2 04		LDX#	04	I/O buffer.
20 FF 03		JSR	DSPHEX	
B1 3E		LDAIY	CRNTAL	Offset?
F0 05		BEQ	DONE	
A2 1C		LDX#	1C	Show offset.
20 FF 03		JSR	DSPHEX	
60	DONE	RTS		

08A1 Subroutine OUTLIN. Output line from IOBUF.

20 2F 1E		JSR	CRLF	New line.
A2 00		LDX#	00	
B5 00	NXTCHR	LDAZX	IOBUF	Output one
20 A0 1E		JSR	OUTCH	character at
E8		INX		a time,
E0 23		CPX#	23	until done.
30 F6		BMI	NXTCHR	
60		RTS		

08B1 Subroutine PRNTCK. Check that FIRST and LAST are legal line numbers. Print lines in range if PRNTOK=1.

A9 00		LDA#	00	Initialize.
85 3E		STAZ	CRNTAL	
A5 40		LDAZ	MDLADL	
85 57		STAZ	OBJECT	
A5 41		LDAZ	MDLADH	
85 58		STAZ	OBJCT1	
A2 07		LDX#	07	Decode range.
20 E1 03		JSR	HX2BIN	
86 59		STXZ	FIRST	
A2 0B		LDX#	0B	
20 E1 03		JSR	HX2BIN	
86 5A		STXZ	LAST	
A9 02		LDA#	02	Initialize flag
85 39		STAZ	WRONG	for mismatch.
20 D9 07	NXTLIN	JSR	DECODE	Decode line.
A5 3E		LDAZ	CRNTAL	
C5 59		CMPZ	FIRST	Decrement WRONG
D0 02		BNE	SKIP	each time a
C6 39		DECZ	WRONG	match is found.
C5 5A	SKIP	CMPZ	LAST	
D0 02		BNE	SKIP2	
C6 39		DECZ	WRONG	
C5 59	SKIP2	CMPZ	FIRST	In range
30 12		BMI	LOW	for print?
C5 5A		CMPZ	LAST	
10 0D		BPL	HIGH	
24 38		BITZ	PRNTOK	Yes, but
30 08		BMI	NOPRNT	print wanted?
A2 1F		LDX#	1F	Yes; add
20 FF 03		JSR	DSPHEX	line number.
20 A1 08		JSR	OUTLIN	Print line.
EA	NOPRNT	NOP		
EA	HIGH	NOP		
18	LOW	CLC		Update pointers.
A5 57		LDAZ	OBJECT	
65 2F		ADCZ	BYTES	
85 57		STAZ	OBJECT	
90 02		BCC	NOINC	
E6 58		INCZ	OBJCT1	
18	NOINC	CLC		
A5 3E		LDAZ	CRNTAL	
65 2F		ADCZ	BYTES	
85 3E		STAZ	CRNTAL	
C5 3D		CMPZ	PRGLEN	Last line?
30 C3		BMI	NXTLIN	If not, repeat.
60		RTS		

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
60	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.

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

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
D0 02		BNE	NOTLST	legal line
C6 39		DECZ	WRONG	number.
A5 39	NOTLST	LDAZ	WRONG	
F0 03		BEQ	OK	
A9 25		LDA#	25	"%" Error-
60		RTS		illegal address.
A5 59	OK	LDAZ	FIRST	
85 3E		STAZ	CRNTAL	
A6 5A		LDXZ	LAST	Deletion needed?
F0 26		BEQ	DONE	
38		SEC		Fix addresses
E5 5A		SBCZ	LAST	for labels.
85 2F		STAZ	BYTES	
20 17 09		JSR	FIXSYM	
A5 3F		LDAZ	CRNTAH	Set pointer
85 5B		STAZ	LAST1	for move.
A5 3D		LDAZ	PRGLEN	Find bytes
38		SEC		to move.
E5 3E		SBCZ	CRNTAL	
85 2D		STAZ	TEMP	
A5 3D		LDAZ	PRGLEN	Correct length
18		CLC		of program.
65 2F		ADCZ	BYTES	
85 3D		STAZ	PRGLEN	
A0 00		LDY#	00	Move lines to
B1 5A	MOVE	LDAIY	LAST	close gap.
91 3E		STAIY	CRNTAL	
C8		INY		
C4 2D		CPYZ	TEMP	
30 F7		BMI	MOVE	
EA		NOP		
A9 2D	DONE	LDA#	2D	"-" Stay in
60		RTS		edit mode.

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

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

Table COMAND. First nine entries in symbol table; commands.

									09B8	3F	41	53	53	47	4E	2E	06
09C0	3F	42	45	47	49	4E	10	06	09C8	2D	4C	4F	43	41	4C	65	06
09D0	3F	52	45	44	45	46	72	06	09D8	2D	41	53	53	45	4D	EB	06
09E0	3F	54	41	42	4C	45	1F	07	09E8	2D	53	54	4F	52	45	A6	07
09F0	2D	50	52	49	4E	54	0D	09	09F8	2D	49	4E	53	52	54	65	09

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.

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

?? 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 4.2: Global Symbols on Page Zero

IOBUF	0000	I/O buffer; prompt or command field.
LABEL	0007	I/O buffer; label field.
OPCODE	000E	I/O buffer; opcode field.
OPRAND	0015	I/O buffer; operand field.
USER	0023	Six bytes available for use by user commands.
ADL	0029	Low address pointer for various subroutines.
ADH	002A	High address pointer.
MISCL	002B	Miscellaneous uses.
MISCH	002C	Ditto.
TEMP	002D	Various temporary uses.
MNE	002E	Mnemonic code.
BYTES	002F	Lengths of lines, etc.
TBL	0030	Low address pointer for table; used by MATCH.
TBH	0031	High address pointer (Subroutine MATCH).
RFL	0032	Low address pointer for string to be matched.
RFH	0033	High address pointer (MATCH).
LEN	0034	Length of each record in table (MATCH).
HBC	0035	Number of highest byte in record which must match.
NUM	0036	Number of highest record in table (MATCH).
OPCPTR	0037	Pointer to opcode in OPCTAB.
PRNTOK	0038	Flag to enable printing by Subroutine PRNTCK.
WRONG	0039	Flag for illegal line numbers (PRNTCK).
MODE	003A	Code for address mode.
SAVX	003B	Used to preserve X register.
GLOBAL	003C	Number of last global symbol.
PRGLEN	003D	Length of source code.
CRNTAL	003E	Low address pointer to current source code line.
CRNTAH	003F	High address pointer.
MDLADL	0040	Module pointer, low address.
MDLADH	0041	Module pointer, high address.
MNETBL	0042	Parameters for MNETAB (see TBL to NUM above).
MODTBL	0049	Parameters for MODTAB.
SYMTBL	0050	Low address pointer to last entry in symbol table.
SYMTBH	0051	High address pointer.
SYMRFL	0052	Low address pointer for symbol to be compared.
SYMRFH	0053	High address pointer.
SYMNUM	0056	Number of last symbol.
OBJECT	0057	Low address pointer to object code.
OBJECT1	0058	High address pointer.
FIRST	0059	First line in range for print (PRNTCK).
LAST	005A	First line after print range.
LAST1	005B	High order address; same as CRNTAH.

Table 4.3: Other Global Symbols

*MNETAB	0200	Three-character ASCII mnemonics for instructions.
*MODTAB	02A8	Two-character ASCII mode codes.
*MIN	02C2	Minimum legal value for MNE for each mode.
*MAX	02CF	Lowest illegal value of MNE for each mode.
*BASE	02DC	Base value for mode added to MNE to get OPCPTR.
*PRMTAB	02E9	Initialization values for CRNTAH through SYMNUM.
*USRPRM	0301	Four bytes available for user parameters.
*OPCTAB	0305	Machine language opcodes pointed to by OPCPTR.
MATCH	03A3	Search table for match to reference.
HEX	03D5	ASCII character to four bits.
HX2BIN	03E1	Two ASCII characters on page zero to eight bits.
BIN2HX	03F2	Four bits to ASCII character on page zero.
DSPHEX	03FF	Eight bits to two ASCII characters, page zero.
SYM	040F	Address of symbol table entry X in MISCL, H.
ADDRSS	0434	Address for symbol X in ADL, H.
ADDLAB	0443	Add symbol to table; return number in X.
NEWSYM	0469	Add symbol if new; call SYM.
ENCODE	047D	Encode assembly language line; update symbols.
CMAND	0549	Look up and transfer control to command.
FIN	0577	Add encoded line to program.
MAIN	05B8	Main program; do command or encode line.
?BEGIN	0610	Add name to symbols; enter edit mode.
?ASSGN	062E	Assign addresses to global symbols.
-LOCAL	0665	Assign addresses to local symbols.
?REDEF	0672	Redefine module pointer.
ASMBL	0683	Translate line into machine code.
LOCSYM	06CB	Display undefined symbols.
-ASSEM	06EB	Assemble module; store at MDL,H.
?TABLE	071F	Reserve space for arrays.
INPUT	075D	Prompt with IOBUF; accept input line.
-STORE	07A6	Save module; clear local symbols; end edit mode.
*MODLIM	07CC	Lower OPCPTR limit for each address mode.
DECODE	07D9	Convert source code to ASCII line.
OUTLIN	08A1	Output line from IOBUF as ASCII.
PRNTCK	08B1	Check line numbers; print lines if enabled.
-PRINT	090D	Output lines in range.
FIXSYM	0917	Revise addresses of symbols in move range.
INSERT	0940	Open gap in source code for insert; fix symbols.
-INSRT	0965	Insert and/or delete lines.

* Table.

[illegible]

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 ?REDEF 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

<u>KIM Routine</u>	<u>Function</u>	<u>Assembler References</u>
CRLF 1E2F	Carriage return, line feed	075D (255D) 08A1 (26A1)
OUTCH 1EA0	Output ASCII from A. Preserve X.	0764 (2564) 08A8 (26A8)
GETCH 1E5A	Input ASCII to A. Preserve X.	0772 (2572)
OUTSP 1E9E	Output one space.	078D (258D)

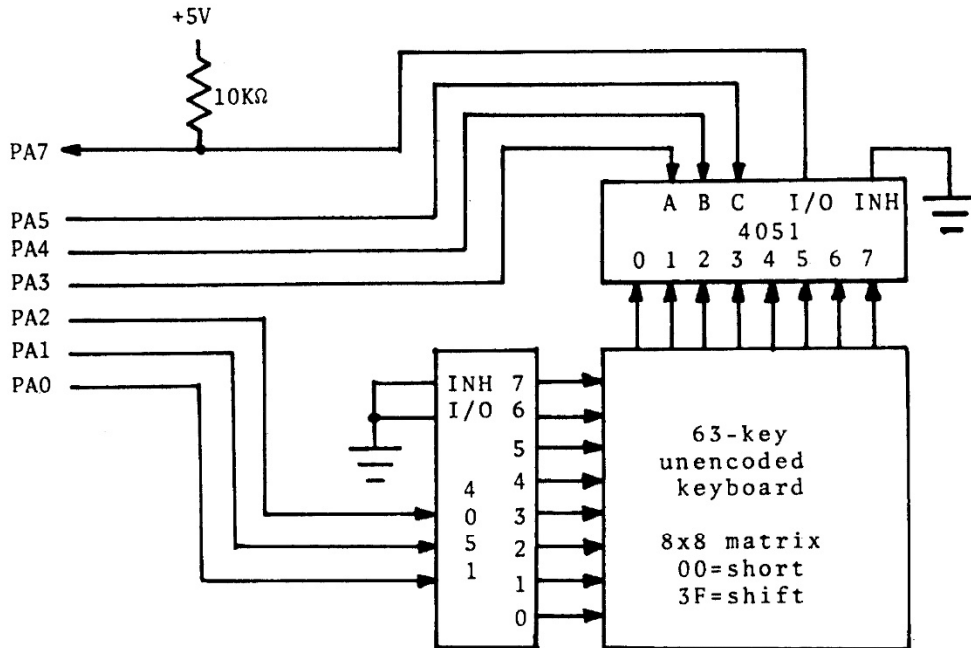


Figure A.1: Keyboard Interface

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 SEGTab and ending with Subroutine CRLF. SEGTab 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		

0EC0 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 7D 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
```

0F00 Subroutine DSPLAY. 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		

OF54 Subroutine ADDCH. Shift ASCII character in A into display from right.

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

OF68 Subroutine GETCH. Get character from keyboard. Return ASCII in A. Add to display or backspace as required. X is preserved.

86 3B		STXZ	SAVX	Save X.
20 00 0F	OLD	JSR	DSPLAY	Wait for release
20 25 0F		JSR	GETKEY	of old key.
D0 F8		BNE	OLD	
EA		NOP		
20 00 0F	NONE	JSR	DSPLAY	Wait for new
20 25 0F		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	LDYZX	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		

0F97 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		

0FAA Subroutine OUTSP. Output one space.

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

0FB0 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, 089C(269C)=1B, 08AD(26AD)=20, 08ED=1E. 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	MDLADH	00
A202		LDX#	02	02
20FF03		JSR	DSPHEX	04
A540		LDAZ	MDLADL	07
A204		LDX#	04	09
20FF03		JSR	DSPHEX	0B
20A108		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	1E
BDE00B	RESTR	LDAX	COPY	20
953C		STAZX	GLOBAL	23
CA		DEX		25
10F8		BPL	RESTR	26
4CD605		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 0681=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 09E0: 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 0A00 through 0C7F.

To retrieve:

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

[illegible]

Permission is hereby granted to photocopy this page.

2K SYMBOLIC ASSEMBLER VERSION 1.0 - SYM USERS' GROUP

Begin session with G 5B8. Block checksum: 0405

```

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0200 42 52 4B 43 4C 43 43 4C 44 43 4C 49 43 4C 56 44,85 BRKCLCCLDCLICLVD
0210 45 58 44 45 59 49 4E 58 49 4E 59 4E 4F 50 50 48,68 EXDEYINXINYNOPPH
0220 41 50 48 50 50 4C 41 50 4C 50 52 54 49 52 54 53,42 APHPPLAPLPRTIRTS
0230 53 45 43 53 45 44 53 45 49 54 41 58 54 41 59 54,09 SECSEDSEITAXTAYT
0240 53 58 54 58 41 54 58 53 54 59 41 43 50 58 53 54,20 SXTXATXSTYACPXST
0250 58 4C 44 58 43 50 59 4C 44 59 53 54 59 41 44 43,FD XLDXCPYLDYSTYADC
0260 41 4E 44 43 4D 50 45 4F 52 4C 44 41 4F 52 41 53,9C ANDCMPEORLDAORAS
0270 42 43 53 54 41 41 53 4C 4C 53 52 52 4F 4C 52 4F,68 BCSTAASLLSRROLRO
0280 52 44 45 43 49 4E 43 42 49 54 4A 4D 50 4A 53 52,15 RDECINCBITJMPJSR
0290 42 43 43 42 43 53 42 45 51 42 4D 49 42 4E 45 42,7C BCCBCSBEQBMBIBNEB
02A0 50 4C 42 56 43 42 56 53 20 20 41 20 23 20 5A 20,3C PLBVCBVS A # Z
02B0 5A 58 5A 59 49 58 49 59 20 20 20 20 58 20 59 20,55 ZXZYIXIY X Y
02C0 49 20 00 27 19 19 1D 1A 1F 1F 30 19 1D 1B 2E 19,54 I ' 0
02D0 2B 26 2E 2D 1C 27 27 38 30 2D 27 2F 00 F2 04 11,5C +&.- '80-' / r
02E0 22 35 32 3A 31 50 63 75 6E 0C 80 0C A5 02 0E 00,33 "52:1Pcun %
02F0 03 02 37 C0 02 11 00 02 01 0C F8 09 15 00 08 05,74 7a x

```

```

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0300 08 FF FF FF FF 00 18 D8 58 B8 CA 88 E8 C8 EA 48,AC XX8J hHjH
0310 08 68 28 40 60 38 F8 78 AA A8 BA 8A 9A 98 0A 4A,A8 h(2`8xx*( : J
0320 2A 6A E0 FF A2 C0 A0 FF 69 29 C9 49 A9 09 E9 E4,3F *j`2 i)II) id
0330 86 A6 C4 A4 84 65 25 C5 45 A5 05 E5 85 06 46 26,71 &D% e%EE% e F&
0340 66 C6 E6 24 B4 94 75 35 D5 55 B5 15 F5 95 16 56,83 fFf$4 u5UU5 u V
0350 36 76 D6 F6 B6 96 61 21 C1 41 A1 01 E1 81 71 31,71 6uUv6 a!AA! a q1
0360 D1 51 B1 11 F1 91 90 B0 F0 30 D0 10 50 70 EC 8E,51 QQ1 q 0p0P Pp1
0370 AE CC AC 8C 6D 2D CD 4D AD 0D ED 8D 0E 4E 2E 6E,E3 .L, m-MM- m N.n
0380 CE EE 2C 4C 20 BC FF 7D 3D DD 5D BD 1D FD 9D 1E,78 Nn,L (<=]] = }
0390 5E 3E 7E DE FE BE FF FF 79 39 D9 59 B9 19 F9,D8 ^)~^^)y9YY9 y
03A0 99 6C FF 86 29 A2 00 86 2A A0 06 B1 29 99 30 00,26 l )" * 1) 0
03B0 88 10 F8 A6 36 A4 35 B1 30 D1 32 F0 02 A0 FF 88,68 x&6$510Q2p
03C0 10 F5 C8 D0 01 60 38 A5 30 E5 34 85 30 B0 02 C6,B9 uHP `8%0e4 00 F
03D0 31 CA 10 E1 60 B5 00 C9 40 30 03 38 E9 07 29 0F,56 1J a`5 120 8i )
03E0 60 20 D5 03 0A 0A 0A 85 2D E8 20 D5 03 05 2D,9A ` U -h U -
03F0 AA 60 C9 0A 30 03 18 69 07 18 69 30 95 00 60 48,20 *`I 0 i i0 `H

```

```

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0400 4A 4A 4A 4A 20 F2 03 E8 68 29 0F 20 F2 03 60 38,92 JJJJ r hh) r `8
0410 86 2D A5 56 E5 2D 85 2B A9 00 85 2C 18 A0 02 26,3C -%Ve- +) , &
0420 2B 26 2C 88 10 F9 38 A5 50 E5 2B 85 2B A5 51 E5,12 +&, y8%Pe+ +%Qe
0430 2C 85 2C 60 20 0F 04 A0 06 B1 2B 85 29 C8 B1 2B,56 , , 1+ >H1+
0440 85 2A 60 85 29 A9 00 85 2A 18 A5 50 69 08 85 50,BE *` )) * %Pi P
0450 90 02 E6 51 A0 07 A9 FF 91 50 88 88 B1 29 91 50,82 fQ ) P 1) P
0460 88 10 F9 A6 56 E8 86 56 60 85 52 A2 50 20 A3 03,C2 y&Vh V` R"P #
0470 F0 05 A5 52 20 43 04 20 0F 04 E4 56 60 A2 42 20,E6 p %R C dV`"B
0480 A3 03 F0 03 A9 31 60 86 2E A2 49 20 A3 03 F0 03,11 # p >1` ."I # p
0490 A9 32 60 A5 2E C9 19 10 02 A2 00 C9 30 30 02 A2,82 >2`%.I " 100 "
04A0 08 EA A5 2E DD C2 02 10 03 A9 33 60 DD CF 02 30,15 j%.1B >3`10 0
04B0 03 A9 33 60 18 7D DC 02 85 37 AA BD 05 03 C9 FF,BA >3` >\ 7*= I
04C0 D0 03 A9 33 60 EA A5 37 C9 1D 10 03 A9 2D 60 E6,A4 P >3`j%7I )-`f
04D0 2F C9 2A 10 0A A2 15 20 E1 03 86 38 A9 2D 60 A2,31 /I* " a 8)-`
04E0 15 86 52 C9 61 10 20 A2 50 20 A3 03 F0 03 A9 34,00 RIa "P # p >4
04F0 60 20 34 04 F0 03 A9 35 60 86 38 A5 1C C9 20 F0,41 ` 4 p >5` 8% I p

```

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
0500	03	A9	36	60	A9	2D	60	EA	A2	50	20	A3	03	F0	05	A9,F9	>6`)-\j"P # p)
0510	15	20	43	04	86	38	A5	37	C9	69	10	0A	E4	3C	10	03,8E	C 8%7Ii d<
0520	A9	37	60	A9	2D	60	EA	E4	3C	30	0A	20	34	04	C5	3F,A4	>7`)-\jd<0 4 E?
0530	D0	03	A9	38	60	A5	1C	A2	00	C9	20	F0	05	A2	1C	20,D7	P >8`% " I p "
0540	E1	03	86	39	E6	2F	A9	2D	60	A5	3A	C5	00	F0	04	18,75	a 9f/)-\%:E p
0550	69	0C	60	A9	00	85	52	A2	50	20	A3	03	F0	0C	A5	00,23	i `) R"P # p %
0560	C9	3F	10	03	A9	30	60	A9	41	60	A9	05	48	A9	75	48,1D	I? >0`>A`) H)uH
0570	20	34	04	6C	29	00	60	20	40	09	A4	2F	88	B9	37	00,1E	4 1) ` @ \$/ 97
0580	91	3E	88	10	F8	A5	07	C9	20	F0	10	A9	07	20	69	04,4F	> x% I p) i
0590	A0	07	A5	3F	91	2B	88	A5	3E	91	2B	18	A5	3E	65	2F,4C	%? + %> + %>e/
05A0	85	3E	18	A5	3D	65	2F	85	3D	10	03	A9	42	60	24	56,37	> %e/ = >B`\$U
05B0	50	03	A9	43	60	A9	2D	60	D8	A2	18	BD	E9	02	95	3F,1A	P)C`)-\X" =i ?
05C0	CA	10	F8	A9	3F	85	00	A0	20	A2	21	94	01	CA	10	FB,46	J x)? " ! J (
05D0	A2	3F	C9	3F	10	10	A5	3F	A2	02	20	FF	03	A5	3E	A2,7E	"?I? %?" %>"
05E0	04	20	FF	03	A2	2D	86	3A	A9	01	85	2F	20	5D	07	A5,BA	"- :> / J %
05F0	3A	C9	2D	D0	04	A5	01	C9	20	D0	0C	20	7D	04	C9	2D,C0	:I-P % I P) I-

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
0600	D0	03	20	77	05	A2	00	F0	03	20	49	05	18	90	B6	EA,7A	P w " p I 6j
0610	A9	07	20	69	04	F0	03	A9	44	60	86	3C	A9	00	85	3E,25	> i p)D` <) >
0620	85	3D	A0	06	91	2B	A5	3F	C8	91	2B	A9	2D	60	A5	07,93	= +%?H +)-\%
0630	C9	20	D0	03	A9	3F	60	A9	07	20	69	04	F0	03	A9	44,B4	I P)?`) i p)D
0640	60	A2	0E	20	E1	03	A0	07	8A	91	2B	A2	10	20	E1	03,6B	`" a "+" a
0650	88	8A	91	2B	A9	20	A2	0C	95	07	CA	10	FB	20	5D	07,A5	+>) " J (J
0660	A5	07	10	CC	EA	20	2E	06	C9	44	D0	03	A9	3A	60	A9,37	% Lj . IDP):`)
0670	2D	60	A2	07	20	E1	03	86	41	A2	09	20	E1	03	86	40,AD	-`" a A" a @
0680	A9	3F	60	A0	00	B1	3E	AA	BD	05	03	91	57	E0	1D	10,E8	)?` 1>*= W`
0690	01	60	C8	B1	3E	E0	2A	10	03	91	57	60	86	2E	AA	20,E3	`H1>`* W` .*
06A0	34	04	A5	29	A0	01	A6	2E	E0	61	10	03	91	57	60	E0,DA	4 %) &.`a W`
06B0	69	10	09	38	E9	02	38	E5	3E	91	57	60	18	C8	71	3E,B1	i 8i 8e> W` Hq>
06C0	88	91	57	C8	A5	2A	69	00	91	57	60	A6	3C	E8	20	34,87	WH%*i W`<h 4
06D0	04	C9	FF	D0	11	A0	05	B1	2B	99	00	00	88	10	F8	86,64	IP 1+ x
06E0	2B	20	A1	08	A6	2B	E4	56	30	E3	60	20	CB	06	A9	2D,9D	+ ! &+dV0c` K)-
06F0	C5	00	F0	01	60	A9	00	85	3E	A5	40	85	57	A5	41	85,4B	E p `) >%@ W%A

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
0700	58	20	83	06	84	2D	38	A5	57	65	2D	85	57	90	02	E6,17	X -8%We- W f
0710	58	38	A5	3E	65	2D	85	3E	C5	3D	30	E5	A9	2D	60	A5,D1	X8%>e- >E=0e)-\%
0720	07	C9	20	D0	03	A9	3F	60	A9	07	20	69	04	F0	03	A9,B5	I P)?`) i p)
0730	44	60	A0	06	A5	40	91	2B	C8	A5	41	91	2B	A2	0E	20,DA	D` %@ +H%A +"
0740	E1	03	8A	18	65	40	85	40	90	02	E6	41	A9	20	A2	0C,FA	a e@ @ fA) "
0750	95	07	CA	10	FB	20	5D	07	A5	07	10	C5	EA	20	4D	83,4A	J (J % Ej M
0760	A2	00	B5	00	20	47	8A	E8	E0	06	30	F6	A2	00	A9	06,D7	" 5 G h` 0v")
0770	85	2D	20	1B	8A	C9	1B	D0	01	00	C9	0D	D0	01	60	C9,D3	- I P I P `I
0780	08	D0	05	CA	E6	2D	A9	08	C9	20	D0	0D	EA	20	42	83,D3	P Jf-) I P j B
0790	E8	C6	2D	10	F8	A9	06	85	2D	C9	20	30	05	95	00	E8,B2	hF- x) -I 0 h
07A0	C6	2D	18	90	CD	EA	A6	3C	20	0F	04	86	56	A5	2B	85,4A	F- MJ&< V%+
07B0	50	A5	2C	85	51	A0	07	A5	41	91	2B	88	A5	40	91	2B,B3	P%, Q %A + %@ +
07C0	18	65	3D	85	40	90	02	E6	41	A9	3F	60	00	19	1D	2A,93	e= @ fA)?` *
07D0	3F	4F	51	59	61	69	80	90	9C	A9	01	85	2F	A2	22	A9,0C	?DQYai) /"")
07E0	20	95	00	CA	10	FB	A6	56	20	34	04	A5	3E	C5	29	D0,8B	J (&V 4 %>E)P
07F0	04	A5	3F	C5	2A	D0	0C	A0	05	B1	2B	99	07	00	88	10,F7	%?E*P 1+

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
0800	F8	A2	01	CA	E4	3C	10	E0	A0	00	B1	57	A2	00	20	FF,D5	x" Jd< \ 1W"
0810	03	B1	3E	85	37	A2	0C	C9	1D	10	02	A2	01	DD	CC	07,7C	1> 7" I " 1L
0820	30	04	86	3A	A2	00	CA	10	F4	A5	3A	0A	AA	BD	A8	02,DA	0 : " J t%: *=(
0830	85	11	BD	A9	02	85	12	B1	3E	38	A6	3A	FD	DC	02	85,D6	=) 1>8&:)\
0840	2D	0A	18	65	2D	AA	BD	00	02	85	0E	BD	01	02	85	0F,07	- e-*= =
0850	BD	02	02	85	10	A5	37	C9	1D	10	01	60	E6	2F	A0	01,46	= %7I \f/
0860	B1	57	A2	02	20	FF	03	A5	37	C9	2A	10	08	B1	3E	A2,8C	1W" %7I* 1>"
0870	15	20	FF	03	60	B1	3E	AA	20	0F	04	A0	05	B1	2B	99,09	\1>* 1+
0880	15	00	88	10	F8	A5	37	C9	69	10	01	60	E6	2F	A0	02,E4	x%7Ii \f/
0890	B1	57	A2	04	20	FF	03	B1	3E	F0	05	A2	1C	20	FF	03,78	1W" 1>p "
08A0	60	20	4D	83	A2	00	B5	00	20	47	8A	E8	E0	23	30	F6,21	\ M " 5 g h`#0v
08B0	60	A9	00	85	3E	A5	40	85	57	A5	41	85	58	A2	07	20,3A	\> %2 W%A X"
08C0	E1	03	86	59	A2	0B	20	E1	03	86	5A	A9	02	85	39	20,17	a Y" a Z) 9
08D0	D9	07	A5	3E	C5	59	D0	02	C6	39	C5	5A	D0	02	C6	39,B9	Y %>EYP F9EZP F9
08E0	C5	59	30	12	C5	5A	10	0D	24	38	30	08	A2	1F	20	FF,C9	EY0 EZ \$80 "
08F0	03	20	A1	08	EA	EA	18	A5	57	65	2F	85	57	90	02	E6,65	! jj %We/ W f

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
0900	58	18	A5	3E	65	2F	85	3E	C5	3D	30	C3	60	A9	01	85,93	X %>e/ >E=0C\)
0910	38	20	B1	08	A9	2D	60	A6	56	20	34	04	C5	3F	D0	1A,1C	8 1 >-&V 4 E?P
0920	A5	29	C5	3E	30	13	A4	29	C4	5A	10	06	A9	FE	A0	07,7F	%>E>0 \$>DZ)~
0930	91	2B	18	65	2F	A0	06	91	2B	EA	CA	E4	3C	10	DA	60,67	+ e/ +jJd< Z\
0940	A5	3E	C5	3D	D0	01	60	85	5A	20	17	09	18	A5	3E	65,FC	%>E=P \ Z %>e
0950	2F	85	29	A5	3F	85	2A	A5	3D	38	E5	3E	A8	B1	3E	91,D1	/ >%? %>=8e>(1>
0960	29	88	10	F9	60	A9	FF	85	38	20	B1	08	C5	5A	D0	02,1A	> y\> 8 1 EZP
0970	C6	39	A5	39	F0	03	A9	25	60	A5	59	85	3E	A6	5A	F0,C9	F9%9p %>%Y >&Zp
0980	26	38	E5	5A	85	2F	20	17	09	A5	3F	85	5B	A5	3D	38,38	&8eZ / %? [%=8
0990	E5	3E	85	2D	A5	3D	18	65	2F	85	3D	A0	00	B1	5A	91,99	e> -%>= e/ = 1Z
09A0	3E	C8	C4	2D	30	F7	EA	A9	2D	60	A2	47	BD	B8	09	9D,DB	>HD-0wj>-\"G=8
09B0	B8	09	CA	10	F7	4C	B8	05	3F	41	53	53	47	4E	2E	06,65	8 J wL8 ?ASSGN.
09C0	3F	42	45	47	49	4E	10	06	2D	4C	4F	43	41	4C	65	06,22	?BEGIN -LOCALe
09D0	3F	52	45	44	45	46	72	06	2D	41	53	53	45	4D	EB	06,D6	?REDEFr -ASSEMk
09E0	3F	54	41	42	4C	45	1F	07	2D	53	54	4F	52	45	A6	07,0A	?TABLE -STORE&
09F0	2D	50	52	49	4E	54	0D	09	2D	49	4E	53	52	54	65	09,05	-PRINT -INSRte

Corrections found by Jeff Tranter

Symbols defined but never used, in table 4.2 on page 43:

USER

RFH

SAVX

MNETBL

MODTBL

SYMRFH

LAST2

Symbols used in the listing but not documented the above table:

IOBUF1

OPCODE1

OPCODE2

OPCODE3

OPCODE4

OFFSET

SYMPTR

OPRDP

Source code errors

Page 16: The two instances of "JSR BIN2HEX" should read "JSR BIN2HX".

Page 24: The third instruction "LDAZ PRMTAB" should be "LDAZX PRMTAB" (or in the more conventional CC65 assembler, "LDA PRMTAB,X").

Page 28: The first instruction "LDA #00" should be "LDY #00".

Page 34: The syntax of some instructions is unusual, e.g. "LDAX MODTAB 01" means "LDAX MODTAB+01" or in the more conventional CC65 assembler, "LDA MODTAB+1,X".

The binary dump at the end of the document was very useful for confirming that the assembled version exactly matched the original.

The only discrepancy in this listing is that the addresses for the I/O routines (GETCH, etc.) do not match the KIM-1 ROM addresses. This is because the dump was for a SYM system as the listing says "SYM USER'S GROUP" (see page 55 of the manual for the changes for a SYM computer).

Note that the last column in the memory dump is a checksum, an 8-bit running total of the bytes in the file up to that point.