

A PAPERBYTE™ BOOK

MONDEB

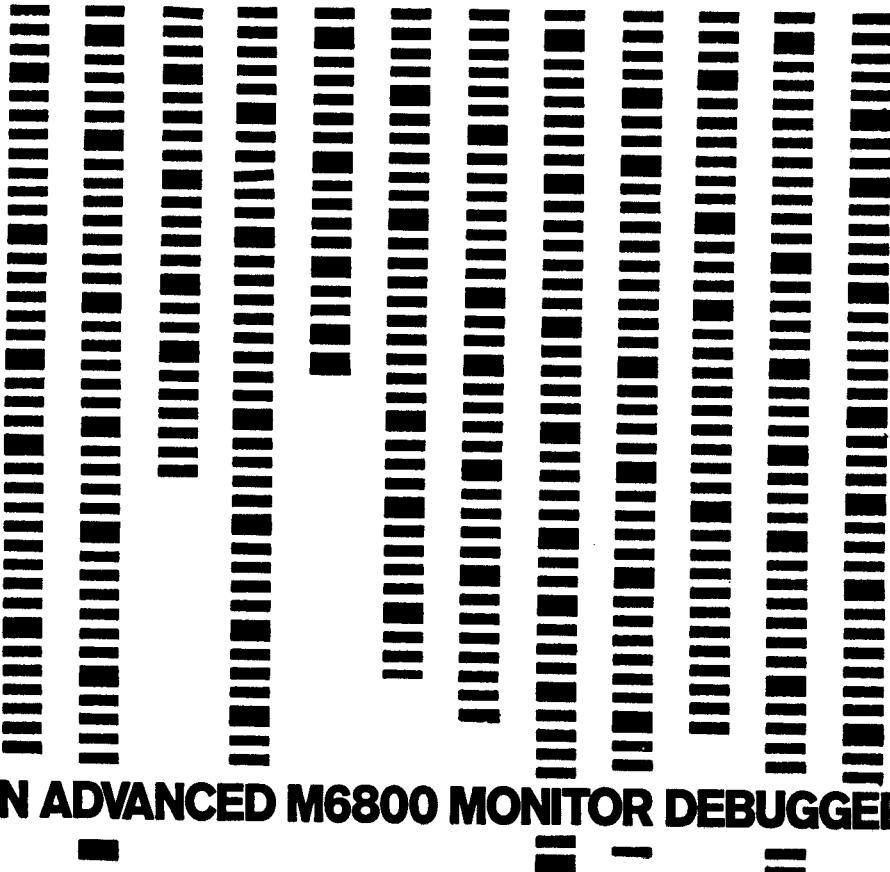
by DON PETERS



AN ADVANCED M6800 MONITOR DEBUGGER

MONDEB

by DON PETERS



AN ADVANCED M6800 MONITOR DEBUGGER

BUTE Publications, Inc.
70 Main Street
Peterborough, New Hampshire 03458

Copyright © 1978 BYTE Publications Inc. All Rights Reserved. BYTE and PAPERBYTE are Trademarks of BYTE Publications Inc. No part of this book may be translated or reproduced in any form without the prior written consent of BYTE Publications Inc.

Published by BYTE Publication, Inc., 70 Main Street, Peterborough, N.H. 03458.

Copyright © 1978 Don Peters. All rights reserved, including the right of reproduction in whole or in part of any form. For information contact BYTE Publications, Inc., 70 Main Street, Peterborough, N.H. 03458.

Peters, Don, 1943

MONDEB-an advanced M6800 monitor-debugger

1. Motorola 6800 (Computer) — Programming. 2. MONDEB (Computer Program) I. Title. QA76.8.M67P47 001.6'425 78-11814
ISBN 0-931718-06-6

Printed in the United States of America

Table of Contents

MONDEB: An Advanced M6800 Monitor-Debugger.....	5
Appendix A: Conversion of TSC BASIC.....	15
Appendix B: MONDEB M6800 Assembly Language Source Listing	17
Appendix C: PAPERBYTE™ Bar Code Representation of MONDEB Object Code.....	79

BYTE Publications Inc. Production Credits

**Blaise Liffick — Technical Editor
Edmond Kelly, Jr. — Production Manager
Walter Banks, University of Waterloo, CCNG, Bar Codes
George Banta Company, Printing
Dawson Advertising, Cover Art**

MONDEB:

An Advanced M6800 Monitor-Debugger

How It Came About

My start in the microcomputer hobby field involved the acquiring of a set of LSI integrated circuits in Motorola's M6800 microprocessor family. It was a truly happy day when, after a long bout of wire wrapping, I applied power, hit the reset button, and the MIKBUG monitor "spoke" to me. Thus began my hardware-software capability spiral which always seemed to consume much more time and money than my initial expectations. While I found the MIKBUG monitor far superior to a front panel crammed with address and data bus switches, its shortcomings became more and more apparent as time went on.

So I began the search for an alternate monitor-debugger. While this kind of software seems to be an important component of any extensive software development effort, this tool has received little attention. This is surprising in light of a recent survey which found that the prime activity of most computer experimenters is software development. While Motorola does offer a few improved 6800 monitors (MINIBUG II, JBUG, etc), these monitors still do not offer significantly increased capability (at least from my point of view), are generally costly, and are not available with source listings. The source code was necessary to fully understand the published description, as my experience with MIKBUG showed. I also have an irresistible urge to customize the software, especially in the areas of input and output. This in turn often requires many program changes, along with reassembly, if one is to avoid numerous messy "hacks" to the code (the sure road to ruin). Turning to various microcomputer magazines, I found descriptions of several monitor-debuggers. But again, they all seemed to be designed as bare bones implementations, sacrificing many conveniences and useful features.

So, not being satisfied with what was available, I decided to make use of my previous experience in writing user interface software for commercial timesharing application programs and develop an advanced (relatively, that is) monitor-debugger. The title MONDEB was selected as a somewhat arbitrary but meaningful acronym.

The monitor-debugger described in this book incorporates all the general features in Motorola's MIKBUG monitor as well as numerous other capabilities. While extremely versatile, ease of use was a prime design consideration.

Goals

Two prime project goals were minimum memory

requirements and maximum versatility. Unfortunately, these goals are generally incompatible, so versatility won out. The final size of MONDEB turned out to be 3 K, implemented on three 2708 type ultraviolet erasable read only memories. This allows frequent alterations and enhancements to be made. Although 2708s have been generally expensive up to now, several manufacturers have recently begun to second source them, bringing the price down considerably.

A more secondary goal was to make this monitor-debugger available to a wide audience of users who could perhaps benefit from and improve upon the design.

General Features

The general features in MONDEB are listed below:

- Liberally commented source code.
- A prompt character signifies readiness to accept a command.
- Commands are generally self-explanatory English words.
- Commands may be abbreviated.
- Commands may be modified by succeeding words called modifiers.
- A space or comma separates a modifier from the command and other modifiers.
- "Rubout" may be used to delete the previous character(s).
- Several commands may be placed on one line, separated from one another by a semicolon.
- "Control-C" may be used to abort the line being typed.
- "Control-Z" will repeat the previous command line.
- Lower case alphabetic input is automatically converted to upper case.
- If a syntax error is made, its position on the input line is pointed to.
- Input lines may be 72 characters long.
- If output lines are over 72 characters long, an automatic carriage return and line feed is inserted after the 72nd character (this is called "folding").
- If a space occurs within the last ten characters on an oversize line, folding occurs on that space.
- One extra null character is sent to the terminal for every eight characters in the output line, allowing ample time for carriage return delays.
- The input and display bases may be set to hexadecimal, decimal, octal or binary (display only).

- ACIA input and output from the terminal or any ACIA address.
- Many routines may be externally accessed through addresses, independent of revision number, decreasing the memory requirements for the application programs.

Command Summary

In the following command summary:

- Capital letters are typed as is.
- Bold faced characters represent the *minimum* abbreviation of a command.
- Lower case text within "<" and ">" represents a variable quantity.
- Text within "[" and "]" is optional.
- An exclamation mark separates alternatives.
- " . . ." represents repetition of the preceding pattern.

REG

```
SET <address><value> [<value>...]
SET <address range><value>
SET .<register><value>
DISPLAY <address range> [DATA ! USED]
DBASE [?! HEX ! DEC ! OCT ! BIN]
IBASE [?! HEX ! DEC ! OCT]
GOTO [<address>]
BREAK [?! <address>]
CONTINUE
TEST <address range>
VERIFY [<address range>]
SEARCH <address range><value> [<value>...]
COPY <address range><address>
COMPARE <value1><value2>
DUMP <address range> [TO <address>]
LOAD [FROM <address>]
DELAY <value>
INT <address>
NMI <address>
SWI <address>
SEI
CLI
```

Command Description

Whenever MONDEB is waiting for a line of input, it prompts with an asterisk (*). When finished typing the line of input, the user types a carriage return and MONDEB begins processing that line. Until the carriage return is typed, the line can be aborted by typing Control-C, or one or more preceding characters can be deleted by typing one or more rubouts. There are two exceptions to this. One is that the first character typed (after the prompt) may be a Control-Z. This will cause the prior line of input to be used for the current line as well. The other exception is that several logical lines may be put on one physical line by separating one logical line from another by a semi-colon (;). Any number of spaces may surround this semi-colon.

In the descriptions that follow, an address range is often called for. This range may be specified as "a:b" which means "address a through address b", or "a!b" which means "starting at address a and for b more bytes." For example, both 100:103 and 100!3 imply the addresses 100, 101, 102, and 103.

Note also that all commands could conceivably be abbreviated to the point where ambiguity sets in. For example, R, RE or REG might each indicate the "display registers" command. Therefore as noted earlier, those characters necessary to uniquely distinguish the command are in bold type in the following descriptions.

REG

The REG command is used to display the contents of the internal registers, as in the following example:

```
*REG
.CC=3C .B=FF .A=23 .IX=1234 .PC=0156 .SP=70A4
```

The period preceding the register name is used to distinguish its name from an ordinary hexadecimal number.

SET

The SET command is used to set the content of memory or the internal registers to specified data. Example:

```
*SET 150 2 10 AA FF
```

This example sets memory location 150 to 2, location 151 to 10, 152 to AA and location 153 to FF. Note that all values are hexadecimal.

If several locations are to be set, it is sometimes useful to "continue" a line with a line feed (LF). This will cause a typeout on the following line of the next address to be set. Simply continue typing input data. Terminate the last byte with a carriage return (CR). In the following example, which illustrates the line feed mode of data entry, the control characters CR (carriage return) and LF (line feed) are shown surrounded by a gray screen:

```
*SET 100 0 1 2 3 (LF)
0104 4 5 6 (LF)
0107 7 (CR)
```

When more than one memory location is to be set to the same value, specify a range with the SET command. For example, to set locations 100 thru 200 to hexadecimal 3F, enter:

```
*SET 100:200 3F
```

The address range in this form of the SET command may only be followed by one data byte.

The internal registers may be set as in the following example:

```
*SET .A 27 .B FF .PC 1234
```

This causes register A to be set to 27, B to FF and PC to 1234. Again, all values are hexadecimal. Note that a period must precede the register name to distinguish it from a memory address specified in hexadecimal. Those registers that may be set are:

CC A B IX PC SP

Since these registers correspond to stack locations, they *only* become effective with the issuance of the CONTINUE command. Note also that changing the value of the stack pointer (SP) effectively changes all register values.

DISPLAY

The DISPLAY command is used to display the contents of a memory address range. For example:

```
*DIS 100:104  
0100=01 0101=5A 0102=23 0103=00 0104=FF
```

Lines exceeding 72 characters in length are folded.

For faster displays of memory, the DATA modifier may be used. It causes the output of records of 16 bytes of data per line with the address of the first byte preceding the data, as shown in the following example:

```
*DIS 100:123, DATA  
0100 01 5A 23 00 FF 01 07 21 00 00 14 14 32 67 00 00  
0110 00 00 CE FA AC A5 54 71 39 00 75 88 72 33 11 22  
0120 AA 01 00 31
```

For even faster displays, the USED modifier will cause a period (.) to represent a zero byte and a plus sign (+) to represent a nonzero byte, as in the example below:

```
*DISPLAY 100: 123 USED  
0100 +++ .++++ ..+++.  
0110 ..++++++ .+++++  
0120 ++.+
```

Note that 16 data values per line are printed when the DATA or USED modifiers are specified and the display base is hexadecimal. If the display base is decimal, ten data values per line are output; for octal, eight values per line; and for binary, two values per line. So, the number of values printed per line indicates the base of the numbers.

DBASE

This command sets the display base to HEX (hexadecimal), DEC (decimal), OCT (octal) or BIN (binary). If no modifier follows this command, HEX is assumed. The following example illustrates this command:

```
*DIS 104  
0104=80  
*DBASE OCT  
*DIS 104  
000404=200  
*DBASE BIN
```

```
*DIS 104  
0000000100000100=10000000  
*DBASE  
*DIS 104  
0104=80
```

Note that the memory address as well as the value is translated to the desired base, but that the input conversion is still hexadecimal in each case (see IBASE, below).

If there is any doubt as to which display base is in effect, follow the DBASE command with a question mark, as:

```
*DBASE ?  
OCT
```

The base in effect will be typed on the succeeding line.

IBASE

Similar in function to the above DBASE command, IBASE is used to set the input base to HEX, (hexadecimal), DEC (decimal), or OCT (octal). Its format is the same as the DBASE command, including the question mark option. Its only difference is that it operates on input values instead of output values.

GOTO

The GOTO command is used to transfer control to a specified memory address, as:

```
*GO 103
```

The address specified is saved so that typing the GOTO command at some future time without a following address value will cause transfer to the last given GOTO address.

BREAK

The BREAK command is used to set a breakpoint at a specified memory address. This is done by replacing the content of the specified address with 3F (hexadecimal), which indicates a "software interrupt" (SWI) instruction. The original content is saved. This saved code is restored when the BREAK command is typed without an address. For example:

```
*BR 763
```

will put an SWI instruction at location 763. Subsequently, typing

```
*BR
```

will remove the SWI code and restore the original instruction.

Upon encountering an SWI instruction, MONDEB will type "SWI:", automatically execute the REG command, and transfer control to command level. The debugged program could be continued, perhaps after exercis-

ing some MONDEB commands, by typing the CONTINUE command, providing that the most recent breakpoint has been removed.

If a breakpoint is set while a prior breakpoint is in effect, the prior breakpoint is automatically removed, ie: only one breakpoint can be set and in effect at a time.

To display the current breakpoint, type BREAK followed by a question mark, as shown in the following example:

```
*BREAK ?
NOT SET
*BR 123
*BR ?
SET @ 0123
```

CONTINUE

This command is used to continue a program that has been interrupted via a breakpoint inserted SWI instruction. Execution will continue at the address of the SWI instruction. Therefore, it is assumed that the SWI instruction at that address has been removed by entering

```
*BREAK
```

alone to restore the former instruction, or by resetting the breakpoint at some other location.

The CONTINUE command also causes the set register command to become effective.

TEST

The TEST command is used to test a programmable memory range for bad memory locations. The test is a simple one in that each location within the range is checked to see if it can store all zeros and then all ones. The addresses of faulty locations and the associated contents are typed out after the check has been completed.

It should be noted that the initial content of the memory location being tested is preserved. Thus, the TEST command doesn't alter memory, making it possible to test memory already loaded with a program or data.

An example of the test command follows. Note that memory locations above hexadecimal FFF are undefined.

```
*TEST 800:1002
1000=00 CANT SET TO ONES
1001=00 CANT SET TO ONES
1002=00 CANT SET TO ONES
1003=00 CANT SET TO ONES
```

VERIFY

The VERIFY command is used to initially compute the checksum of a memory range, and then subsequently to compare this reference checksum to a new one generated for the same address range, as shown below:

```
*VERIFY 0:FFF
3C
*VER
OK
```

```
*SET 0013 23
*VER
CHECKSUM ERROR
```

This command is useful when checking out new software to insure that some unforeseen bug has not caused it to destroy part of itself.

The Motorola MIKBUG definition of the checksum of a range is simply the complement of the sum of all the bytes in the range.

SEARCH

SEARCH is used to search a memory range for a specified string of bytes. When the sequence is found, the address of the first matching byte is displayed. The maximum length of a search string is six bytes. Note that the locations of all matching strings in the search range are typed, not just the first.

A good use for this command is in program conversion, where, for instance, all jumps to a certain subroutine must be changed to another subroutine, as in input and output conversions. Example:

```
*SEARCH 800:FFF BD FD 06
```

Note that the address range (800:FFF) is followed by the byte string (BD FD 06, hexadecimal) being searched for.

COPY

The COPY command is used to copy a range of bytes from one memory location to another. The source range is followed by the start of the destination range, as shown in the example below:

```
*COPY 750!4 20
```

Note that the copy will not work properly if the source range partly overlays the destination range *and* if the first address of the destination range exceeds the first address of the source range. In other works, you can shift a range down a few bytes, but not up.

COMPARE

The COMPARE command simply types out the sum and difference between two specified numbers. This eases the burden of mental computations in nondecimal bases. For example, when trying to patch in a BSR instruction, the relative difference of two addresses may be needed, as in:

```
*COMPARE 53F 4FF
SUM IS 0A3E, DIF IS 0040
```

Note that in subtraction, the second number is subtracted from the first.

INT

This command allows you to define the location to

which control will transfer upon receipt of an interrupt other than a nonmaskable or software interrupt. For example:

*INT 6074

NMI

Similar to the INT command, NMI defines the location to which control will transfer upon receipt of a nonmaskable interrupt.

SWI

The SWI command is also similar to the INT command, but defines the location to which control will transfer upon encountering a software interrupt (SWI) instruction.

SEI

SEI sets the interrupt mask bit, causing interrupts to be ignored. It has no modifiers.

CLI

CLI clears the interrupt mask bit, causing subsequent interrupts to be processed normally. CLI also has no modifiers.

DUMP

DUMP provides a way to save a portion of memory on paper tape or cassette tape. The format of the dumped data is identical to that of the Motorola MIKBUG monitor, except that header (type S0) and trailer (type S9) records are also included. The example of this command:

*DUMP 600:2400

will dump the address range 600 thru 2400, inclusive. If an address range is not given, the range stored in RANGLO and RANGHI (see source listing on pages 35 and 36) is used. This provides the capability of having an external program set up this range for a subsequent DUMP by MONDEB.

By default, the dumped information will go to the user's terminal. The dump may be sent to another device (such as a cassette) if that device is interfaced through an ACIA. Use the optional parameter TO keyword followed by the address of the desired ACIA to dump the information to some device other than the terminal. For example, a paper tape punch might be interfaced through an ACIA whose data address is 7F45 (and control address is 7F44). To dump memory locations 1000 through 2000 to the paper tape punch, the following command would be used:

*DUMP 1000:2000 TO 7F45

Leaders consisting of 30 null characters precede the first record and follow the last record.

Note that the display base should be set to HEX if the dump is to be a normal Motorola MIKBUG hexadecimal dump.

LOAD

Motorola MIKBUG formatted tapes can be loaded with the LOAD command. To load from cassette tapes, simply type the LOAD command with no modifiers.

To load from another ACIA controlled device, append "FROM" and the data address of the ACIA receiving the formatted load information, as:

*LOAD FROM 7F41

DELAY

The DELAY command will delay the prompt for (and processing of) the next line of input for the specified number of milliseconds. This feature is intended for the testing of peripheral devices. It possibly attains its greatest value when interspersed with several other commands on a composite input line creating delays between the commands. This is done where timing of an event is crucial. For example, the following could be used to send three characters to an ACIA controlled remote terminal at 256 millisecond intervals:

*SET 7F45 30;DELAY 100;SET 7F45 31;DELAY
100;SET 7F45 32

Note that the values are all hexadecimal.

The delay is generated by an internal loop. The loop time in turn is dependent upon the microprocessor clock rate and a preset variable at memory location TIMCON. This variable should be set to 100 for a 1 MHz clock, or 50 for a 0.5 MHz clock. If your processor clock runs at X MHz, then the closest integer value for TIMCON is found by rounding the result of the following expression:

TIMCON = 100 * X

Coding Conventions

Good coding conventions simplify the problems of software interface and modification. It is with this in mind that this monitor adheres to the following conventions:

- MONDEB resides in the upper 3 K of a 6800's memory address space (61 K to 64 K) leaving all lower memory available for user memory. See figure 1 for a map layout of the MONDEB memory space.
- Page zero (addresses 0 to 255) is not used at all, eliminating possible storage conflicts with application software.
- The scratch pad memory required for MONDEB resides at about 30 K, but a reassembly could place it almost anywhere.
- Monitor routines for reading and displaying a char-

acter or character string are included.

- Register A is generally used to pass 1 byte data to and from MONDEB. Other registers are generally preserved.
- Jump vectors in high memory define the entry points to frequently used subroutines so that subsequent

MONDEB revisions will not affect the access addresses of dependent software.

- A standard Motorola MIKBUG formatted DUMP and LOAD utility is available.
- The source code listing is in standard Motorola format.

Figure 1: Map of MONDEB memory space.

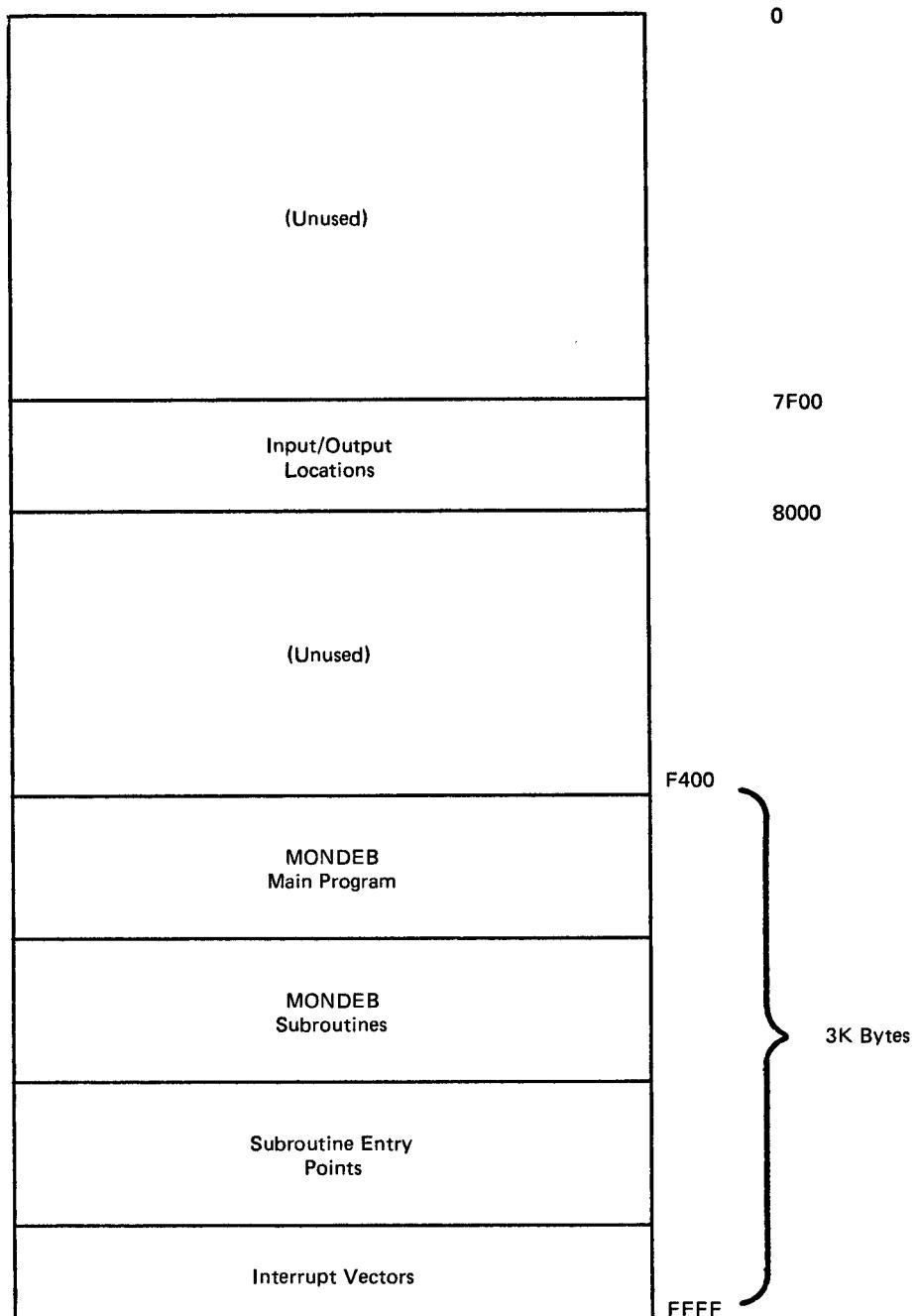


Figure 2: Example of scanning of a line of input.

Assembly Language		Accumulator A Input	Accumulator A Returned	NBRHI	NBRLO	Analysis Pointer
JSR	GETLIN					
LDX	BUFBEG					
STX	SYNPTR	—	—			↑SET 100 2A
LDA A #1		1	11			
JSR	COMAND	(Search list 1)	(11th command)	—	—	SET ↑100 2A
LDA A #2		1	-1			
JSR	COMAND	(Search list 1)	(No match)	—	—	SET ↑100 2A
JSR	NUMBER	—	+1	01	00	SET 100 ↑2A
JSR	NUMBER	—	+1	00	2A	SET 100 2A ↑
JSR	NUMBER	—	0	00	00	SET 100 2A ↑

Input Line Scanning

The nucleus of MONDEB consists of a number of routines which scan an input line for certain information. Each routine looks for its own specific item of interest and signals success or failure depending upon whether or not the desired item is found. For example, the routine NUMBER looks for a number, ie: a string of digits in the expected base. The result of the scan, or search, is a number which is returned in a predetermined memory location.

The concept of a line or syntax pointer is central to the scanning process. This pointer simply identifies the actual location within the input line at any moment during the scanning process. This pointer is called SYNPTR in the program, and is initialized to the beginning (first character) of the input line (input buffer) when an input line is requested via subroutine GETLIN. At this point the input line is scanned to a point where MONDEB can decide that it has a particular item, such as a number or command, located on the line. Let us return for a moment to the example of scanning for a number.

If a good number were scanned, the pointer would be advanced *through* the sequence of numeric digits, stopping at the next terminator (a nonnumeric character). The pointer would then be in position to pick up the next item on the input line (after skipping over intervening delimiters). However, if the number were not valid (for instance, if 1Z3 were scanned), the pointer would not be advanced at all, allowing an alternate routine to take over. This might be the case if, for example, a number or a command could syntactically appear in the scanned position.

All this leads to the necessity of determining the status of the scan. The convention of the line scanning routines is to store the scan status in accumulator A. If accumulator A is positive, the scan was successful, and the pointer advances to the first item delimiter. If it is zero, then the end of an input line has been reached (nothing

else to scan), and the pointer is moved to the end of the line. If accumulator A is negative, the scan was unsuccessful, and the pointer remains unaltered.

Since the scanning routines set up the A accumulator, just prior to their exit, they also set up the condition code Z and N bits for testing after the subroutine call, further simplifying the scan status tests.

The following example illustrates the above, using MONDEB (with all of MONDEB's editing facilities available) to get a number from a terminal:

```

10 JSR    GETLIN  Get a line of input
20 LDX    BUFBEG Get address of beginning of buffer
30 STX    SYNPTR Get syntax pointer equal to it
40 JSR    NUMBER Scan for the number
50 BMI    ERROR   To ERROR if bad number
60 BEQ    EOLINE  To EOLINE if end-of-line

```

Here, line 10 gets an input line (with all editing features such as rubout, Control-Z, Control-C, etc) to process. Lines 20 and 30 set the scan pointer to the start of this input line. Line 40 looks for a valid number and sets up the condition code bits appropriately. Line 50 then checks for an error and branches to the label ERROR if a number is not recognized. Line 60 causes a branch to label EOLINE if there is nothing more on the line to scan. If a good number is scanned, the most significant byte of the number would be stored in NBRHI and the least significant byte in NBRLO. To pick up a second number, lines 40, 50 and 60 would be repeated. See figure 2 for an example of scanning a line.

Any good interactive software should be able to handle key words with the same ease as numeric scanning. MONDEB offers this facility through a subroutine called COMAND. COMAND operates on lists of key words. For example, one such list in MONDEB consists of HEX, DEC, OCT and BIN. Each list is associated with a number, and the words in the previous example happen to be (arbitrarily)

assigned to list #3.

The following example, instead of looking for numeric data on the input line, seeks a match on some word (command) within list #3:

```
10 LDA A #3      Refer to list 3
20 JSR COMMAND   Look for a match
30 BLT ERROR     To ERROR if no match
40 BEQ EOLINE    To EOLINE if end-of-line
```

Note the similarity to the NUMBER routine; the only thing extra needed is the list in which to search for a match. If a match is found, accumulator A contains the positional number of the matched command. For example, if DEC is scanned, accumulator A would be set to 2, since DEC is the second entry in the list. Note that since COMAND permits abbreviations, the user could have scanned for DEC, DE or simply D. Again, remember that the pointer is not advanced on a match failure, so another match on another list can be tried.

Setting up these command lists is also fairly easy. They can be entered as text strings with the FCC assembler directive, following each command with a carriage return character, and terminating each list with a line feed character. This is shown in the MONDEB assembly listing. Different lists can be set up in the initialization section of the code section, which could be accomplished as follows:

```
10 LDX #LISTS-1
20 STX COMADR
```

In this example, LISTS is the label of the first command in the first list of the command lists. COMADR is a memory location that always holds the address of the start of the command lists it is to use.

While NUMBER and COMAND are the two subroutines of major importance, there are many other useful routines. All, including the above two, are documented below.

MONDEB Subroutine Summary

Routine name: TIMDEL.
Entry address: FFB9.
Description: Execute a time delay.
Input: The index register specifies the number of milliseconds to delay.
Output: None.
Register preserved: Accumulator B.

Routine name: CKSUM.
Entry address: FFBC.
Description: Compute the checksum of an address range.
Inputs: This routine and the input address storage locations are used by the VERIFY command.
VERFRM holds the address of the beginning of the range.
VERTO holds the address of the end of the range.

Output:	Accumulator A holds the computed checksum.
Register preserved:	Accumulator B.
Routine name:	GETCHR.
Entry address:	FFBF.
Description:	Reads a character from memory (input line).
Input:	LINPTR holds the address preceding the address of the character to get.
Output:	Accumulator B holds the character read.
	LINPTR is left pointing to the address the character came from.
Register preserved:	Accumulator A.
Routine name:	GETLST.
Entry address:	FFC2.
Description:	Reads a character from memory (command list).
Input:	LISPTR holds the address preceding the address of the character to get.
Output:	Accumulator A holds the character read.
	LISPTR is left pointing to the address the character came from.
Register preserved:	Accumulator B.
Routine name:	GTRANG.
Entry address:	FFC5
Description:	Scan for a pair of numbers. A ":" separating the pair implies "thru". A separating "!" implies "thru the following," eg: 100:105 is equivalent to 100:15. A single number is valid and gets put into both of the range high and range low address storage locations.
Input:	(input line).
Outputs:	(RANGLO, RANGLO+1) holds the range start. (RANGHI, RANGHI+1) holds the range end.
Registers preserved:	None.
Routine name:	NUMBER.
Entry address:	FFC8.
Description:	Scan for a 1 or 2 byte number in the input base currently in effect.
Input:	(input line).
Outputs:	Accumulator A is negative: Illegal number, pointer not advanced. Accumulator A is zero: End of line reached, pointer set there. Accumulator A is positive: Valid number scanned, pointer advanced to next delimiter. NBRHI: High byte of scanned number. NBRLO: Low byte of scanned number.
Register preserved:	Index register.

Routine name:	SKPDLM.	Routine name:	TYPCMD.
Entry address:	FFCB.	Entry address:	FFD7.
Description:	Skip over leading delimiters until a nondelimiter or end-of-line character is found.	Description:	Types out a given command in a given list.
Input:	(input line).	Inputs:	Accumulator A holds the command list number.
Output:	The carry bit is set if an end of line is encountered.	Outputs:	COMNUM holds the command number within that list.
Registers preserved:	None.	Registers preserved:	None.
Routine name:	TSTDLM.	Routine name:	OUT1BY.
Entry address:	FFCE.	Entry address:	FFDA.
Description:	Test whether specified character is a delimiter.	Description:	Outputs a 1 byte number with leading zeros.
Inputs:	Accumulator B holds the character to be tested. DELIM specifies the delimiter class as follows:	Inputs:	The index register points to the address of the byte to output.
	<p>1 = Only space is a delimiter. 2 = Only comma is a delimiter. 3 = Space or comma is a delimiter. 4 = Any nonalphanumeric character is a delimiter.</p>		DBCODE specifies the output base.
Output:	<p>Accumulator A = 0 : Character is not a delimiter. Accumulator A = 1 : Character is a delimiter.</p>	Output:	<p>DBCODE = 1 : HEXadecimal DBCODE = 2 : DECimal DBCODE = 3 : OCTal DBCODE = 4 : BINary</p>
Registers preserved:	Accumulator B and Index register.	Registers preserved:	Numeric characters.
Routine name:	TSTEOL.	Routine name:	OUT2BY.
Entry address:	FFD1.	Entry address:	FFDD.
Description:	Test for an end-of-line character.	Description:	Outputs a 2 byte number with leading zeros.
Input:	Accumulator A holds the character to be tested.	Input:	The index register holds the address of the most significant byte of the pair of bytes to output.
Output:	The Z bit of the condition code is set if the character in accumulator A is a line terminator, ie: a carriage return (CR), line feed (LF) or semicolon (;).	Output:	DBCODE specifies the output base (see OUT1BY).
Registers preserved:	Accumulator B and index register.	Registers preserved:	Numeric characters.
Routine name:	COMAND.	Routine name:	GETLIN.
Entry address:	FFD4.	Entry address:	FFE0.
Description:	A match is sought on one of the commands in the list specified by accumulator A on input. The result of the match attempt is reflected by accumulator A and the condition code bits N and Z.	Description:	Gets a line of text entered by the user.
Inputs:	(input line).		The line is terminated by entering a carriage return. A carriage return, line feed pair is automatically inserted in the input line after 72 input characters. However, the pair does not get inserted into the input buffer. Exceeding the input buffer length (default is 72 characters) causes the message "TOO LONG" to be typed. The following editing characters are available:
Outputs:	<p>Accumulator A holds the number of the list to be searched. Accumulator A = -1: Match unsuccessful, pointer not advanced. Accumulator A = 0 : End of line, pointer advanced to end of line. Accumulator A = +n : Successful match on command in list position "n." Pointer advanced to command delimiter.</p>		<p>Rubout deletes the previous character. The deleted characters are surrounded by backslashes in the input line echoed to the terminal. Control-C will abort the line. Control-Z (as the first character</p>
Registers preserved:	None.		

of an input line) will use the previous line as the current line.

Input: A line of characters entered by the user.

Output: Characters stored in an input line buffer which have a beginning address stored in BUFBEG and an ending address stored in BUFEND. Accumulator B is set to 3 upon line abort by Control-C.

Registers preserved: None.

Routine name: OUTSTR.

Entry address: FFE3.

Description: Output a character string terminated by a code of 4 (ie: End of Transmission (EOT) or Control-D).

Input: Index register holds the address of the beginning of the string.

Output: Character string to terminal, ACIA or memory (see OUTCHR).

Registers preserved: Accumulator A and accumulator B.

Routine name: DOCRLF.

Entry address: FFE6.

Description: Output a carriage return followed by a line feed to the terminal. Variable CPLCNT (characters-per-line count) is cleared. After the carriage return and line feed, one null character (a "filler") is sent for every 16 characters of line length, allowing time for the hardware to react.

None.

Carriage return and line feed, plus "fillers" to the terminal.

Registers preserved: Accumulator A, accumulator B and index register.

Routine name: OUTCHR.

Entry address: FFE9.

Description: Output a character to the desired output device or address location, as follows:

OUTFLG = 0: Output is to the terminal (via TOACIA).

OUTFLG = 1: Output is to the ACIA data address stored in OUTADR.

OUTFLG = 2: Output is to the address in OUTADR which is then incremented.

Input: Accumulator A holds the character to be output.

Output: One character is sent to a specified destination. If output is to the terminal, a carriage return followed by a line feed is interjected after every 72

characters, or during a space in the last ten characters in a line.

Registers preserved: Accumulator A, accumulator B and index register.

Routine name: TOACIA.

Entry address: FFEC.

Description: Output a character to the terminal.

Input: Accumulator A holds the character to be sent to the terminal.

Output: One character is sent to the terminal.

Registers preserved: Accumulator A, accumulator B and index register.

Routine name: INPCHR.

Entry address: FFEF.

Description: Input a character from an ACIA, as follows:

INPFLG = 0: Input is from the terminal.

INPFLG = 1: Input is from the ACIA data address stored in INPADR.

Input: A character from an ACIA.

Output: Accumulator A returns with the character.

Registers preserved: Accumulator B and index register.

Conclusion

An extensive "wish list" preceded development of MONDEB. As development progressed, this list shrank, but never as fast as new features were implemented. Knowing development would continue virtually forever through "one plussing," the decision was made to interrupt development at the 3 K byte level, a point where most of the important features were felt to be included.

This monitor-debugger was sort of a "bootstrap" project in that it greatly eases the development of future software. Its much greater power compared to MIKBUG makes it a real pleasure to use.

While the project was interesting and challenging, it also took much longer than planned (slipped schedules seem to be the norm in software development). But then again there is the satisfaction of seeing a complex product developed to your own critical specifications.

It is my hope that this monitor-debugger will be as helpful to others as it has been to me, and that this implementation will lead to bigger, better, and more useful versions. ■

Appendix A:

Conversion of TSC BASIC

The following modifications convert Technical Systems Consultants' BASIC to interface to the MONDEB monitor-debugger. Note that, except for the included "to", the syntax of MONDEB's SET command is used to describe the memory changes required.

To get an echo (MONDEB uses full duplex operation):

SET 106 to BD FF EF
SET 109 to 7E FF E9

SET 450 to 01 09
SET 2D1 to 01 06
SET 7BF to 01 06
SET 7FE to 01 06

To disable the test of the MIKBUG PIA for a break:

SET 452 to 39

To have BASIC set MONDEB's GO command to restart BASIC at 103:

SET 1B4 to 70 19

To cause BASIC's MON command to jump to MONDEB's prompt instead of MIKBUG:

SET 15F to FF F2

To set up for use of MONDEB's stack instead of MIKBUG's stack:

SET 1B7 to 70 B1
Set 946 to 70 9C

To have BASIC set MONDEB's DUMP command to dump a BASIC program:

SET 1C4 to 70 15
SET 1C9 to 70 17

To change "delete last character" from Control-H to rubout:

SET 2D4 to 7F

To change "delete line" from Control-X to Control-C:

SET 2E3 to 3
SET 7C2 to 3

Appendix B:

MONDEB M6800 Assembly Language Source Listing

NAM MONDEB

*THIS SOURCE CODE WAS SENT TO WALTER BANKS AT
*THE UNIVERSITY OF WATERLOO BY DON PETERS ON PAPER TAPE
*CROSS ASSEMBLY WAS DONE ON THE U OF W HONEYWELL 66/60
*THE BARCODE AND LISTING WERE SET ON A PHOTON PHOTO-
*TYPESETTER DRIVEN BY THE HONEYWELL.

*

* M O N D E B - A MONITOR/DEBUGGER FOR THE M6800
* MICROPROCESSOR

* AUTHOR: DON PETERS
* DATE: APRIL 1977
* MEMORY REQ'D: 3K BYTES AT HIGH END OF ADDRESS SPACE

* SEE USER MANUAL FOR CAPABILITIES & INSTRUCTIONS ON
* USE

F400 * ORG \$400 DEBUG ORG AT 1K
* ORG \$F400 NORMAL ORIGIN AT 61K

*I/O DEVICE ADDRESSES

7F43 ACIA1 EQU \$7F43 ACIA #1 - MAIN TERMINAL ACIA
7F45 ACIA2 EQU \$7F45 ACIA #2 - AUXILIARY TERMINAL
* ACIA

*OTHER CONSTANTS

000D CR EQU 13 CARRIAGE RETURN
000A LF EQU 10 LINE FEED

F400 START EQU * PROGRAM ENTRY POINT
F400 8E 70B1 LDS #STACK INITIALIZE THE STACK POINTER
F403 BF 7006 STS SP SAVE THE POINTER
F406 BD FE08 JSR INITAL INITIALIZE VARIABLES

*TYPE OUT MONITOR NAME & VERSION

F409 BD FEC7 JSR DOCRLF ADVANCE TO A CLEAN LINE
F40C CE FEF2 LDX #MSGHED GET ADDRESS OF HEADER
F40F BD FE4B JSR OUTSTR TYPE IT

*SET UP DESTINATION OF INPUT LINE

*DEFINE BEGINNING OF INPUT BUFFER

F412 CE 702F LDX #TTYBUF-1 GET ADDRESS OF TERMINAL
* INPUT BUFFER
F415 FF 702C STX BUFBEG SAVE IT

*DEFINE END OF INPUT BUFFER - 72 CHAR CAPACITY, INCL CR

F418 CE 7078 LDX #TTYEND
F41B FF 702E STX BUFEND

*DELIMITER CLASS DEFINITION - SPACE OR COMMA (CODE 3)

F41E 86 03 LDA A #3
F420 B7 700F STA A DELIM
F423 20 0F BRA PROMP1

*PREPARE TO GET A NEW COMMAND

F425 BD FEC7 PROMPT JSR DOCRLF TYPE CR-LF
F428 7C 700E INC BOLFLG SET "BEGINNING OF LINE" FLAG
F42B FE 700A LDX SYNPTR POINT TO CURRENT CHARACTER
F42E A6 00 LDA A X GET IT
F430 81 3B CMP A #' ; SEMICOLON?
F432 27 1A BEQ GETCMD CONTINUE SCAN IF IT IS,
SKIPPING THE PROMPT

*TYPE PROMPT

F434 CE FEFF PROMP1 LDX #MSGPRM
F437 BD FE4B JSR OUTSTR

F43A BD FD8C JSR GETLIN GET LINE OF INPUT

*ABORT LINE ON A CONTROL-C

F43D C1 03 CMP B #3
F43F 27 E4 BEQ PROMPT

*SET SYNTAX SCANNING POINTER TO BEGINNING OF
* BUFFER/LINE

F441 FE 702C LDX BUFBEG
F444 FF 700A STX SYNPTR

*REPROMPT ON AN EMPTY LINE (FIRST CHAR = CR, LF, OR ;)

F447 A6 01 LDA A 1,X GET FIRST CHAR
F449 BD FA89 JSR TSTEOL TEST IT
F44C 27 D7 BEQ PROMPT IF IT IS, PROMPT AGAIN

*USE LIST 1 WHEN MATCHING

F44E 86 01 GETCMD LDA A #1

*NOW GO FOR A MATCH

F450 BD F9C7 JSR COMAND

*-AND TEST THE RESULT OF THE SCAN

F453 27 D0 BEQ PROMPT REPROMPT IF JUST A CR WAS TYPED
F455 2E 1F BGT JMPCMD GOOD COMMAND IF POSITIVE

*UNRECOGNIZABLE SYNTAX - POINT TO ERROR

F457 FE 702C BADSYN LDX BUFBEG GET START OF LINE
*SPACE OVER TO ERROR IN SYNTAX

F45A BC 700C BADS1 CPX LINPTR AT ERROR?

F45D 27 06	BEQ	BADS2	
F45F BD FBF1	JSR	OUTSP	OUTPUT A SPACE
F462 08	INX		NO, MOVE ON
F463 20 F5	BRA	BADS1	
*THE "EXTRA" CHAR "1" IS COMPENSATED FOR BY THE PROMPT			
*CHAR ON THE PRECEEDING LINE			
F465 86 SE	BADS2	LDA A # '1	AT ERROR - GET AN UP-ARROW
F467 BD FE76	JSR	OUTCHR	PRINT IT
F46A BD FEC7	JSR	DOCRLF	
F46D 20 C5	BRA	PROMP1	IGNORE ANY SUCCEEDING PACKED
*COMMANDS			

*THERE SHOULD BE NO MORE CHARACTERS ON THE INPUT LINE			
*(EXCEPT DELIMITERS)			
F46F BD FA69	NOMORE	JSR SKPDLM	
F472 25 B1		BCS PROMPT	IF CARRY BIT SET, END OF LINE
*			
(NORMAL)			
F474 20 E1		*THERE IS SOMETHING THERE BUT SHOULDN'T BE	
		BRA BADSYN	

*EXECUTE A COMPUTED "GOTO" TO THE PROPER COMMAND			
F476 16	JMPCMD TAB		SAVE COMMAND # IN ACCB
F477 48		ASL A	MULTIPLY COMMAND BY 2
F478 1B		ABA	ACCA NOW HOLDS COMMAND #
*			
MULTIPLIED BY 3			
*ADD IT TO BASE OF JUMP TABLE			
F479 C6 F4		LDA B #JMPHI	GET HI BYTE OF START OF JUMP
TABLE IN ACCB			
F47B 8B 85		*	ADD A #JMPLO
ADD LO BYTE OF START OF JUMP			
TABLE TO ACCA			
F47D C9 00		*	ADC B #0
ADD CARRY IF THERE WAS			
ONE			
*MOVE ACCA & ACCB TO IX (CODE IS WEIRD, BUT BRIEF)			
F47F 36		PSH A	
F480 37		PSH B	
F481 30		TSX	PUT ADDRESS OF "GOTO" INTO X
F482 EE 00		LDX X	GET THE ADDRESS ITSELF
F484 33		PUL B	RESTORE THE STACK
F485 32		PUL A	
F486 6E 00 JMP X JUMP TO RIGHT COMMAND			
F485	JMPTBL	EQU	* - 3
00F4	JMPHI	EQU	JMPTBL / 256
F400	JMP256	EQU	JMPHI * 256
0085	JMPLO	EQU	JMPTBL - JMP256
F488 7E F4C7	JMP	REG	
F48B 7E F514	JMP	GOTO	
F48E 7E F526	JMP	SEI	
F491 7E F529	JMP	CLI	

F494	7E	F52C	JMP	COPY
F497	7E	F558	JMP	BREAK
F49A	7E	F5B8	JMP	I BASE
F49D	7E	F5CE	JMP	DBASE
F4A0	7E	F604	JMP	CONTIN
F4A3	7E	F608	JMP	DISPLA
F4A6	7E	F673	JMP	SET
F4A9	7E	F720	JMP	VERIFY
F4AC	7E	F766	JMP	SEARCH
F4AF	7E	F7ED	JMP	TEST
F4B2	7E	F844	JMP	INT
F4B5	7E	F84C	JMP	NMI
F4B8	7E	F854	JMP	SWI
F4BB	7E	F85C	JMP	COMPAR
F4BE	7E	F885	JMP	DUMP
F4C1	7E	F924	JMP	LOAD
F4C4	7E	F9B5	JMP	DELAY

*REG - DISPLAY REGISTERS				
F4C7	REG	EQU	*	
*PRINT STACK STORED SWI DATA				
F4C7	FE	7006	DISREG LDX SP	GET SAVED STACK POINTER
F4CA	08		INX	
*REGISTER NAME TYPEOUT INITIALIZATION				
F4CB	7F	70D6	CLR COMNUM	START AT BEGINNING OF THE
	*			REGISTER NAME LIST
F4CE	8D	13	BSR OUT2	TYPE CONDITION CODES
F4D0	8D	11	BSR OUT2	TYPE ACCB
F4D2	8D	0F	BSR OUT2	TYPE ACCA
F4D4	8D	14	BSR OUT4	TYPE INDEX REG
F4D6	8D	12	BSR OUT4	TYPE PROGRAM COUNTER
*TYPE THE STACK POINTER LOCATION				
F4D8	8D	18	BSR OUT2A4	TYPE STACK POINTER ID
F4DA	CE	7006	LDX #SP	
F4DD	BD	FC04	JSR OUT2BY	TYPE THE VALUE
F4E0	7E	F46F	JMP	NOMORE
*OUTPUT CONTENT OF A 1 BYTE REGISTER				
F4E3	8D	0D	OUT2 BSR OUT2A4	
F4E5	BD	FBFD	JSR OUT1BY	
F4E8	08		INX	
F4E9	39		RTS	
*OUTPUT CONTENT OF A 2 BYTE REGISTER				
F4EA	8D	06	OUT4 BSR OUT2A4	
F4EC	BD	FC04	JSR OUT2BY	
F4EF	08		INX	SKIP TO NEXT BYTE IN STACK
F4F0	08		INX	SKIP TO NEXT BYTE IN STACK

F4F1 39

RTS

*MISC SETUP FOR REGISTER DISPLAY

F4F2 BD FBF1	OUT2A4	JSR	OUTSP	OUTPUT A SPACE
F4F5 7C 70D6		INC	COMMUN	SKIP TO NEXT REGISTER NAME
F4F8 86 05		LDA A	#5	REGISTER NAME IS IN LIST 5
F4FA BD FA2E		JSR	TYPCMD	TYPE IT
F4FD BD FBF7		JSR	OUTEQ	TYPE AN "="
F500 39		RTS		

*ENTER HERE FROM SOFTWARE INTERRUPT

F501 CE FF01	TYPSWI	LDX	#MSGWSI	
F504 BD FE4B		JSR	OUTSTR	
				*DECREMENT PC SO IT POINTS TO "SWI" INSTRUCTION
F507 FE 7006		LDX	SP	
F50A 6D 07		TST	7,X	TEST LO BYTE OF PC FOR PENDING BORROW
F50C 26 02		BNE	TYPSWI	
F50E 6A 06		DEC	6,X	NEED TO BORROW, DECR HI BYTE OF PC
F510 6A 07	TYPSWI	DEC	7,X	DECR LO BYTE OF PC
F512 20 B3		BRA	DISREG	GO DISPLAY REGISTERS

*GOTO - GO TO MEMORY ADDRESS

F514 BD FB47	GOTO	JSR	NUMBER	GET DESTINATION
F517 27 08		BEQ	GOTO1	IF NONE, USE DEFAULT
F519 FE 7013		LDX	NBRHI	
F51C FF 7019		STX	LASTGO	SAVE IT
F51F 6E 00		JMP	X	GO TO DESTINATION

F521 FE 7019	GOTO1	LDX	LASTGO	GET LAST GOTO ADDRESS
F524 6E 00		JMP	X	GO TO IT

*SEI - SET INTERRUPT MASK

F526 0F	SEI	SEI		
F527 20 2C		BRA	COPY3	

*CLI - CLEAR INTERRUPT MASK

F529 0E	CLI	CLI		
F52A 20 29		BRA	COPY3	

*COPY - COPY FROM ONE LOCATION TO ANOTHER

F52C BD FAFC	COPY	JSR	GTRANG	GET SOURCE RANGE INTO RANGLO & RANGHI
	*			

F52F 2F 21		BLE	COPY2	ERROR IF NO SOURCE
------------	--	-----	-------	--------------------

F531 BD FB47		JSR	NUMBER	GET DESTINATION
--------------	--	-----	--------	-----------------

F534 2F 1C		BLE	COPY2	ERROR IF NO DESTINATION
------------	--	-----	-------	-------------------------

F536 FE 7015		LDX	RANGLO	GET SOURCE ADDRESS POINTER
--------------	--	-----	--------	----------------------------

F539 A6 00	COPY1	LDA A	X	GET BYTE FROM SOURCE
------------	-------	-------	---	----------------------

F53B FE 7013		LDX	NBRHI	GET DESTINATION ADDRESS POINTER
--------------	--	-----	-------	---------------------------------

F53E A7 00		STA A	X	SAVE BYTE IN DESTINATION
------------	--	-------	---	--------------------------

F540 08	INX		INC DESTINATION POINTER
F541 FF 7013	STX	NBRHI	SAVE IT
F544 FE 7015	LDX	RANGLO	GET SOURCE ADDRESS POINTER
F547 BC 7017	CPX	RANGHI	COMPARE TO END OF INPUT RANGE
F54A 27 09	BEQ	COPY3	DONE IF EQUAL
F54C 08	INX		NOT EQUAL, INC SOURCE POINTER
F54D FF 7015	STX	RANGLO	SAVE IT
F550 20 E7	BRA	COPY1	LOOP FOR NEXT BYTE

F552 7E F457 COPY2	JMP	BADSYN	BAD SYNTAX
F555 7E F46F COPY3	JMP	NOMORE	SHOULD BE NO MORE ON THE INPUT LINE
*			

*BREAK - SET BREAKPOINT AT SPECIFIED ADDRESS & REMOVE OLD ONE

*			
F558 BD FB47 BREAK	JSR	NUMBER	GET BREAKPOINT LOCATION
F55B 2B 31	BMI	BREAK3	IF NOT NUMERIC, LOOK FOR "?"
F55D 27 1F	BEQ	BREAK2	IF NO MODIFIER, REMOVE OLD BREAKPOINT
*			

*CHECK IF A "SWI" IS STORED AT THE BREAK ADDRESS

F55F FE 7020	LDX	BRKADR	GET CURRENT BREAK ADDRESS
F562 A6 00	LDA A	X	AND THE CHAR THERE
F564 81 3F	CMP A	#\$3F	COMPARE TO "SWI"
F566 26 05	BNE	BREAK1	EQUAL?
*			
*YES, RESTORE THE OLD INSTRUCTION			
F568 B6 7022	LDA A	BRKINS	GET IT
F56B A7 00	STA A	X	RESTORE IT

*PUT BREAK AT NEWLY SPECIFIED LOCATION

F56D FE 7013 BREAK1	LDX	NBRHI	GET NEW BREAKPOINT (BREAK ADDRESS)
*			
F570 FF 7020	STX	BRKADR	SAVE IT
F573 A6 00	LDA A	X	GET INSTRUCTION STORED THERE
F575 B7 7022	STA A	BRKINS	SAVE IT
F578 86 3F	LDA A	#\$3F	GET CODE FOR SOFTWARE INTERRUPT
F57A A7 00	STA A	X	PUT IT AT BREAKPOINT
F57C 20 34	BRA	BREAK5	ALL DONE

*REMOVE BREAKPOINT

F57E FE 7020 BREAK2	LDX	BRKADR	GET ADDRESS OF BREAK
F581 A6 00	LDA A	X	GET INST. THERE
F583 81 3F	CMP A	#\$3F	SWI?
F585 26 2B	BNE	BREAK5	IF NOT, RETURN & PROMPT
F587 B6 7022	LDA A	BRKINS	WAS A SWI - GET PREVIOUS INST.
F58A A7 00	STA A	X	& RESTORE IT
F58C 20 24	BRA	BREAK5	

*LOOK FOR A QUESTION MARK IN LIST 4

F58E 86 04 BREAK3	LDA A	#4	
F590 BD F9C7	JSR	COMAND	SCAN FOR IT
F593 2F 20	BLE	BREAK6	BAD SYNTAX IF NOT "?"

F595 FE 7020	LDX BRKADR	IT IS, GET BREAK ADDRESS
F598 A6 00	LDA A X	GET INSTRUCTION THERE
F59A 81 3F	CMP A #\$3F	IS IT A "SWI"?
F59C 27 08	BEQ BREAK4	IF YES, SAY SO

*NO BREAKPOINT SET

F59E CE FF10	LDX #MSGNBR	GET THAT MESSAGE
F5A1 BD FE4B	JSR OUTSTR	SAY IT
F5A4 20 0C	BRA BREAK5	

*BREAKPOINT SET

F5A6 CE FF18	BREAK4 LDX #MSGBAT	GET THAT MESSAGE
F5A9 BD FE4B	JSR OUTSTR	SAY IT
F5AC CE 7020	LDX #BRKADR	GET BREAK ADDRESS
F5AF BD FC04	JSR OUT2BY	TYPE IT

F5B2 7E F46F	BREAK5 JMP	NOMORE
F5B5 7E F457	BREAK6 JMP	BADSYN

*IBASE - SET INPUT BASE

*LOOK FOR HEX, DEC, OR OCT IN LIST #3

F5B8 86 03	IBASE LDA A #3	
F5BA BD F9C7	JSR COMAND	
F5BD 2B 09	BMI IBASE2	UNRECOGNIZABLE BASE, TRY "?"
F5BF 2E 02	BGT IBASE1	
F5C1 86 01	LDA A #1	NO BASE GIVEN - DEFAULT TO HEX
F5C3 B7 7010	IBASE1 STA A IBCODE	SAVE BASE CODE
F5C6 20 EA	BRA BREAK5	

*LOOK FOR "?" IN LIST #4

F5C8 B6 7010	IBASE2 LDA A IBCODE	GET IB CODE IN CASE ITS NEEDED
F5CB 36	PSH A	SAVE IT ON STACK TEMPORARILY
F5CC 20 24	BRA DBASE4	

*DBASE - SET DISPLAY BASE

*LOOK FOR HEX, DEC, OCT, OR BIN IN LIST #3

F5CE 86 03	DBASE LDA A #3	
F5D0 BD F9C7	JSR COMAND	
F5D3 2B 19	BMI DBASE3	UNRECOGNIZABLE BASE, TRY "?"
F5D5 2E 02	BGT DBASE1	
F5D7 86 01	LDA A #1	NO BASE GIVEN - DEFAULT TO HEX
F5D9 B7 7011	DBASE1 STA A DBCODE	

*COMPUTE THE NUMERIC DISPLAY BASE (FOR THE "DISPLAY" COMMAND)

F5DC CE F5E9	LDX #DBTBL-1	POINT TO HEAD OF DISPLAY BASE TABLE
F5DF 08	DBASE2 INX	INC TABLE POINTER
F5E0 4A	DEC A	DECR DISPLAY BASE CODE
F5E1 26 FC	BNE DBASE2	LOOP IF NOT EQUAL
F5E3 A6 00	LDA A X	EQUAL - GET NUMERIC BASE FROM TABLE
F5E5 B7 7012	STA A DBNBR	SAVE IT
F5E8 20 C8	BRA BREAK5	DONE

*DISPLAY BASE TABLE

FSEA 10	DBTBL	FCB	16
FSEB 0A		FCB	10
FSEC 08		FCB	8
FSED 02		FCB	2

*LOOK FOR "?" IN LIST #4

FSEE B6 7011	DBASE3	LDA A	DBCODE	GET DB CODE IN CASE ITS NEEDED SAVE IT ON STACK TEMPORARILY
--------------	--------	-------	--------	--

F5F1 36		PSH A		
---------	--	-------	--	--

F5F2 86 04	DBASE4	LDA A	#4	
------------	--------	-------	----	--

F5F4 BD F9C7		JSR	COMAND	
--------------	--	-----	--------	--

F5F7 33		PUL B		RETRIEVE INPUT BASE/DISPLAY BASE CODE
---------	--	-------	--	--

*

F5F8 2F BB		BLE	BREAK6	ERROR IF THE "SOMETHING" WAS
------------	--	-----	--------	------------------------------

*

*SET UP FOR TYPEOUT OF BASE CODE

F5FA 86 03		LDA A	#3	ITS IN LIST 3
------------	--	-------	----	---------------

F5FC F7 70D6		STA B	COMNUM	STORE BASE CODE
--------------	--	-------	--------	-----------------

F5FF BD FA2E		JSR	TYPCMD	TYPE OUT BASE
--------------	--	-----	--------	---------------

F602 20 AE		BRA	BREAK5	
------------	--	-----	--------	--

*CONTINUE - CONTINUE FROM A "SWI"

*RETURN TO LOCATION WHERE SWI WAS

F604 BE 7006	CONTIN	LDS	SP	IN CASE SP WAS MODIFIED VIA SET COMMAND
--------------	--------	-----	----	--

F607 3B		RTI		
---------	--	-----	--	--

*DISPLAY - DISPLAY MEMORY DATA

F608 BD FAFC	DISPLA	JSR	GTRANG	GET MEMORY DISPLAY RANGE
--------------	--------	-----	--------	--------------------------

F60B 2F 60		BLE	DISPL9	ADDRESS IS REQUIRED
------------	--	-----	--------	---------------------

*INITIALIZE ADDRESS POINTER TO START OF MEMORY

F60D FE 7015		LDX	RANGLO	
--------------	--	-----	--------	--

F610 FF 70B8		STX	MEMADR	
--------------	--	-----	--------	--

*SEARCH LIST 6 FOR DISPLAY MODIFIERS "DATA" OR "USED"

F613 86 06		LDA A	#6	
------------	--	-------	----	--

F615 BD F9C7		JSR	COMAND	
--------------	--	-----	--------	--

F618 2B 53		BMI	DISPL9	ANY OTHER MODIFIER IS ILLEGAL
------------	--	-----	--------	-------------------------------

*ADJ DISPLAY MODIFIER CODE SO THAT: -1=ADDR & DATA,
* 0=DATA, 1=USED

F61A 4A		DEC A		
---------	--	-------	--	--

F61B B7 70D6		STA A	COMNUM	SAVE FOR LATER TESTS
--------------	--	-------	--------	----------------------

*INIT "DATA VALUES PER LINE" COUNTER

F61E 5F		CLR B		
---------	--	-------	--	--

F61F 5C		INC B		
---------	--	-------	--	--

F620 CE 70B8	DISPL1	LDX	#MEMADR	
--------------	--------	-----	---------	--

F623 7D 70D6		TST	COMNUM	WHICH DISPLAY OPTION?
--------------	--	-----	--------	-----------------------

F626 2B 2C		BMI	DISPL6	IF "ADDRESS & DATA", GO THERE
------------	--	-----	--------	-------------------------------

*OUTPUT DATA WITH ADDRESS ONLY AT LINE BEGINNING

F628 5A		DEC B		COUNT DATA VALUES PER LINE
---------	--	-------	--	----------------------------

F629 26 0C		BNE	DISPL2	IF COUNT NOT UP, SKIP ADDRESS OUTPUT
------------	--	-----	--------	---

*

F62B BD FEC7	JSR DOCRLF	GET TO LINE BEGINNING
F62E BD FC04	JSR OUT2BY	OUTPUT ADDRESS
F631 BD FBF1	JSR OUTSP	AND A SPACE
F634 F6 7012	LDA B DBNBR	RESET LINE COUNTER
F637 FE 70B8 DISPL2	LDX MEMADR	POINT TO DATA AT THAT ADDRESS
F63A 7D 70D6	TST COMMUN	WANT "DATA" OPTION?
F63D 2E 05	BGT DISPL3	IF NOT, GO TO "USED" CODE
* "DATA" OPTION		
F63F BD FBF1	JSR OUTSP	OUTPUT PRECEEDING SPACE
F642 20 1B	BRA DISPL7	
* "USED" OPTION		
F644 A6 00	DISPL3 LDA A X	GET THE DATA
F646 4D	TST A	EXAMINE IT FOR ZERO
F647 26 04	BNE DISPL4	
F649 86 2E	LDA A #'	ITS ZERO, GET A ".."
F64B 20 02	BRA DISPL5	
F64D 86 2B	DISPL4 LDA A #' +	ITS NON-ZERO, GET A "+"
F64F BD FE76	DISPL5 JSR OUTCHR	OUTPUT THE "." OR "+"
F652 20 0E	BRA DISPL8	
F654 BD FBF1 DISPL6	JSR OUTSP	OUTPUT A PRECEEDING SPACE
F657 BD FC04	JSR OUT2BY	TYPE ADDRESS
F65A BD FBF7	JSR OUTEQ	TYPE "="
F65D EE 00	LDX X	GET CONTENT
F65F BD FBFD	DISPL7 JSR OUT1BY	TYPE IT
F662 BC 7017 DISPL8	CPX RANGHI	ARE WE DONE?
F665 27 09	BEQ DISP10	IF YES, BACK TO PROMPT
F667 08	INX	NO, INC MEMORY ADDRESS
F668 FF 70B8	STX MEMADR	SAVE IT
F66B 20 B3	BRA DISPL1	
F66D 7E F457 DISPL9	JMP BADSYN	
F670 7E F46F DISPL10	JMP NOMORE	

*SET - SET MEMORY LOCATIONS		
F673 BD FAFC SET	JSR GTRANG	GET MEMORY LOCATION/RANGE
F676 2B 4E	BMI SETS	IF NOT AN ADDRESS, LOOK FOR A REGISTER NAME
F678 27 F3	BEQ DISPL9	AN ADDRESS MODIFIER IS REQUIRED
*RANGE OF ADDRESSES SPECIFIED?		
F67A FE 7015	LDX RANGLO	
F67D BC 7017	CPX RANGHI	
F680 27 12	BEQ SET2	IF SINGLE ADDRESS, SET UP ADDRESSES INDIVIDUALLY
*		
*SET A RANGE OF ADDRESSES TO A SINGLE VALUE		
F682 BD FB47	JSR NUMBER	GET THAT VALUE

F685 2F E6	BLE	DISPL9	ITS REQUIRED
F687 B6 7014	LDA A	NBRLO	PUT IT IN ACCA
F68A A7 00 SET1	STA A	X	STORE IT IN DESTINATION
F68C BC 7017	CPX	RANGHI	END OF RANGE HIT?
F68F 27 DF	BEQ	DISP10	IF YES, ALL DONE
F691 08	INX		NO, ON TO NEXT ADDRESS IN RANGE
F692 20 F6	BRA	SET1	LOOP TO SET IT
*SET ADDRESSES UP INDIVIDUALLY			
F694 FF 70B8 SET2	STX	MEMADR	SAVE MEMORY LOC
F697 BD FB47 SET3	JSR	NUMBER	GET DATA TO PUT THERE
F69A 27 0D	BEQ	SET4	END OF LINE?
F69C 2D CF	BLT	DISPL9	ABORT IF BAD SYNTAX
F69E B6 7014	LDA A	NBRLO	LOAD DATA BYTE
F6A1 FE 70B8	LDX	MEMADR	LOAD ADDRESS
F6A4 A7 00	STA A	X	STORE DATA

* INCREMENT ADDRESS IN CASE USER WANTS TO INDIVIDUALLY
 * SET SEVERAL

* SUCCESSIVE LOCATIONS

F6A6 08	INX		
F6A7 20 EB	BRA	SET2	

* END OF LINE - WAS IT TERMINATED WITH A LINE FEED?

F6A9 FE 700A SET4	LDX	SYNPTR	POINT TO END OF LINE
F6AC A6 00	LDA A	X	GET CHAR THERE
F6AE 81 0A	CMP A	#LF	LINE FEED?
F6B0 26 6B	BNE	SET12	IF NOT, BACK TO PROMPT
F6B2 CE 70B8	LDX	#MEMADR	YES, GET NEXT ADDRESS TO BE SET
F6B5 BD FC04	JSR	OUT2BY	TYPE IT
F6B8 BD FBF1	JSR	OUTSP	AND A SPACE
F6BB BD FD8C	JSR	GETLIN	GET A NEW LINE
F6BE FE 702C	LDX	BUFBEG	GET BUFFER BEGINNING
F6C1 FF 700A	STX	SYNPTR	EQUATE IT TO SYNTAX SCAN POINTER
F6C4 20 D1	BRA	SET3	GO PICK UP DATA

* LOOK FOR (REGISTER NAME, REGISTER VALUE) PAIRS

F6C6 86 05	SETS	LDA A	#5	
F6C8 BD F9C7		JSR	COMAND	PICK UP A REGISTER NAME
F6CB 2B 4D		BMI	SET11	ERROR IF UNRECOGNIZABLE
F6CD 27 4E		BEQ	SET12	DONE IF END OF LINE
F6CF 36		PSH A		SAVE REGISTER NAME(NUMBER)
F6D0 BD FB47		JSR	NUMBER	GET NEW REGISTER VALUE
F6D3 32		PUL A		RESTORE REGISTER NAME(NUMBER)
F6D4 2F 44		BLE	SET11	GOT GOOD REGISTER VALUE?
F6D6 FE 7006		LDX	SP	YES, POINT TO TOP OF STACK
F6D9 F6 7014		LDA B	NBRLO	GET REGISTER VALUE

* CONDITION CODES

F6DC 81 01	CMP A	#1	
F6DE 26 04	BNE	SET6	
F6E0 E7 01	STA B	1,X	
F6E2 20 E2	BRA	SETS5	

*ACCB

F6E4	81	02	SET6	CMP A	#2
F6E6	26	04		BNE	SET7
F6E8	E7	02		STA B	2,X
F6EA	20	DA		BRA	SET5

*ACCA

F6EC	81	03	SET7	CMP A	#3
F6EE	26	04		BNE	SET8
F6F0	E7	03		STA B	3,X
F6F2	20	D2		BRA	SET5

*IX

F6F4	81	04	SET8	CMP A	#4
F6F6	26	09		BNE	SET9
F6F8	B6	7013		LDA A	NBRHI
F6FB	A7	04		STA A	4,X
F6FD	E7	05		STA B	5,X
F6FF	20	C5		BRA	SET5

UPDATE HI BYTE
UPDATE LO BYTE

*PC

F701	81	05	SET9	CMP A	#5
F703	26	09		BNE	SET10
F705	B6	7013		LDA A	NBRHI
F708	A7	06		STA A	6,X
F70A	E7	07		STA B	7,X
F70C	20	B8		BRA	SET5

UPDATE HI BYTE
UPDATE LO BYTE

*SP

F70E	81	06	SET10	CMP A	#6
F710	26	08		BNE	SET11
F712	FE	7013		LDX	NBRHI
F715	FF	7006		STX	SP
F718	20	AC		BRA	SET5

DON'T NEED IX TO SET SP

F71A	7E	F457	SET11	JMP	BADSYN
F71D	7E	F46F	SET12	JMP	NOMORE

*VERIFY - CHECKSUM VERIFY A BLOCK OF MEMORY

F720	BD	FAFC	VERIFY	JSR	GTRANG	GET A NUMBER RANGE
F723	27	1B		BEQ	VERIFI	NO MODIFIER MEANS CHECK WHAT WE HAVE
	*					
F725	2B	F3		BMI	SET11	ANYTHING ELSE IS ILLEGAL
			*GOOD RANGE GIVEN,			TRANSFER IT TO CHECKSUM ADDRESSES
F727	FE	7015		LDX	RANGLO	
F72A	FF	701B		STX	VERFRM	
F72D	FE	7017		LDX	RANGHI	
F730	FF	701D		STX	VERTO	

F733	8D	22		BSR	CKSUM	COMPUTE CHECKSUM
F735	B7	701F		STA A	CHKSUM	SAVE IT
F738	CE	701F		LDX	#CHKSUM	TYPE THE CHECKSUM
F73B	BD	FBFD		JSR	OUTIBY	

F73E 20 DD

BRA SET12

*NO MODIFIER GIVEN - JUST VERIFY CHECKSUM
F740 8D 15 VERIFI BSR CKSUM COMPUTE CHECKSUM
F742 B1 701F CMP A CHKSUM SAME AS STORED CHECKSUM?
F745 26 08 BNE VERIF2

*THEY VERIFY - SAY SO

F747 CE FF1F LDX #MSGVER
F74A BD FE4B JSR OUTSTR
F74D 20 CE BRA SET12

*THEY DON'T - SAY SO

F74F CE FF22 VERIF2 LDX #MSGNVE
F752 BD FE4B JSR OUTSTR
F755 20 C6 BRA SET12

*COMPUTE THE CHECKSUM FROM ADDRESSES VERFRM TO VERTO
*RETURN THE CHECKSUM IN ACCA

F757 4F CKSUM CLR A INIT CHECKSUM TO ZERO
F758 FE 701B LDX VERFRM GET FIRST ADDRESS
F75B 09 DEX INIT TO ONE LESS
F75C 08 CKSUM1 INX START OF CHECKSUM LOOP
F75D AB 00 ADD A X UPDATE CHECKSUM IN ACCA WITH
* BYTE POINTED TO
F75F BC 701D CPX VERTO HIT END OF RANGE?
F762 26 F8 BNE CKSUM1 IF NOT, LOOP BACK
F764 43 COM A COMPLEMENT THE SUM
F765 39 RTS RETURN WITH IT

*SEARCH - SEARCH MEMORY FOR A BYTE STRING

*GLOBAL VARIABLES USED

*LINPTR - INPUT LINE CHARACTER POINTER
*LISPTR - COMMAND LIST CHARACTER POINTER
*RANGLO - "SEARCH FROM" ADDRESS
*RANGHI - "SEARCH TO" ADDRESS

*LOCAL VARIABLES USED

*MEMADR - STARTING MEMORY ADDRESS WHERE A MATCH
* OCCURRED
* BYTPTR - ADDRESS POINTER USED TO FILL BYTSTR AND
* SUBSTR BUFFERS
*NBYTES - NUMBER OF BYTES IN BYTE STRING
*NBRMAT - NUMBER OF CHARS THAT MATCH SO FAR IN THE
* MATCHING PROCESS
*BYTSTR - STARTING ADDRESS OF 6 CHARACTER BYTE STRING
* BUFFER

*THE SEARCH STRING OCCUPIES TEMP4, TEMP5, & TEMP6 (6
* BYTES MAX)

```

        *GET SEARCH RANGE BEGINNING (RANGLO) & END (RANGHI)
F766 BD FAFC SEARCH JSR      GTRANG
F769 2F 7C           BLE     SEARC9    ABORT IF NO PAIR

        * INITIALIZE BYTE STRING POINTER
F76B CE 70BE         LDX      #BYTSTR   GET START OF BYTE STRING TO
*                           SEARCH FOR
F76E FF 70BA         STX      BYTPTR    SET POINTER TO IT

F771 7F 70BC         CLR      NBYTES   ZERO # OF BYTES IN BYTE STRING

        *GET A BYTE STRING
F774 BD FB47 SEARCI JSR      NUMBER   GET A BYTE
F777 27 IA           BEQ      SEARC2   BEGIN SEARCH IF EOL
F779 2D 6C           BLT      SEARC9

        *GOOD BYTE, ADD IT TO STRING
F77B 7C 70BC         INC      NBYTES   COUNT THIS BYTE
*DON'T ACCEPT OVER 6 BYTES
F77E B6 70BC         LDA A   NBYTES
F781 81 06           CMP A   #6
F783 2E 62           BGT      SEARC9

F785 B6 7014         LDA A   NBRLO   GET (LOW ORDER) BYTE
F788 FE 70BA         LDX      BYTPTR   GET BYTE POINTER
F78B A7 00           STA A   X        SAVE BYTE
F78D 08           INX      MOVE BYTE POINTER TO NEXT
*                           LOCATION IN STRING
F78E FF 70BA         STX      BYTPTR   SAVE IT
F791 20 E1           BRA      SEARCI

        *BEGIN SEARCH FOR BYTE STRING
* IS # OF BYTES TO LOOK FOR >0
F793 7D 70BC SEARC2 TST      NBYTES
F796 27 4F           BEQ      SEARC9   IF NOT, BAD SYNTAX

        *MAKE USE OF INPUT LINE CHARACTER FETCH & COMMAND LIST
*
        *INITIALIZE MEMORY POINTER TO START OF SEARCH RANGE
F798 FE 7015         LDX      RANGLO
F79B 09           DEX
F79C FF 700C         STX      LINPTR

        *INITIALIZE BYTE POINTER TO START OF BYTE STRING
F79F CE 70BD SEARC3 LDX      #BYTSTR-1
F7A2 FF 70D7         STX      LISPTR

F7A5 7F 70BD         CLR      NBRMAT   SET "NUMBER OF BYTES THAT"

```

* MATCHED" TO ZERO
 *GET BYTE FROM BYTE STRING & RETURN IT IN ACCA
 F7A8 BD FCCD JSR GETLST
 *GET BYTE FROM MEMORY RANGE & RETURN IT IN ACCB
 F7AB BD FCC0 SEARC4 JSR GETCHR

F7AE 11 CBA COMPARE MEMORY & BYTE STRING
 * CHARACTERS
 F7AF 27 07 BEQ SEARC5 IF NO MATCH, TEST FOR RANGE END
 F7B1 BC 7017 CPX RANGHI HAVE WE REACHED THE RANGE
 * SEARCH UPPER LIMIT?
 F7B4 27 34 BEQ SEAR10 YES, GO PROMPT FOR NEXT COMMAND
 F7B6 20 F3 BRA SEARC4

*MATCH ACHIEVED - SAVE ADDRESS OF MATCH
 F7B8 FF 70B8 SEARC5 STX MEMADR
 F7BB 7C 70BD SEARC6 INC NBRMAT BUMP NUMBER MATCHED
 F7BE B6 70BD LDA A NBRMAT
 F7C1 B1 70BC CMP A NBYTES HAVE ALL CHARACTERS MATCHED?
 F7C4 27 16 BEQ SEARC8 IF SO, MATCH ACHIEVED
 *HAVEN'T MATCHED ALL YET, GO GET NEXT PAIR EVEN IF
 * PAST "SEARCH TO" ADDRESS

F7C6 BD FCCD JSR GETLST
 F7C9 BD FCC0 JSR GETCHR
 F7CC 11 CBA
 F7CD 27 EC BEQ SEARC6
 *MISMATCH ON SOME BYTE PAST THE FIRST ONE
 *RESET THE MEMORY POINTER TO GET NEXT UNTESTED MEMORY
 * LOCATION

F7CF FE 70B8 SEARC7 LDX MEMADR
 *THIS TEST HANDLES SPECIAL CASE OF A MATCH ON RANGE END
 F7D2 BC 7017 CPX RANGHI
 F7D5 27 13 BEQ SEAR10
 F7D7 FF 700C STX LINPTR
 *GO RESET THE BYTE STRING POINTER
 F7DA 20 C3 BRA SEARC3

*MATCH ON BYTE STRING ACHIEVED, TYPE OUT MEMORY ADDRESS
 F7DC CE 70B8 SEARC8 LDX #MEMADR
 F7DF BD FC04 JSR OUT2BY
 F7E2 BD FBF1 JSR OUTSP AND A SPACE
 *ASSUME A MISMATCH (I.E., RESET MEMORY & BYTE STRING
 * POINTERS & CONTINUE

F7E5 20 E8 BRA SEARC7

F7E7 7E F457 SEARC9 JMP BADSYN
 F7EA 7E F46F SEAR10 JMP NOMORE

* TEST - TEST RAM FOR BAD BYTES
 *GET AN ADDRESS RANGE
 F7ED BD FAFC TEST JSR GTRANG

F7F0 2F F5	BLE	SEARC9	ABORT IF NO PAIR
	* RANGLO HOLDS STARTING ADDRESS OF RANGE		
	* RANGHI HOLDS ENDING ADDRESS OF RANGE		
F7F2 FE 7015	LDX	RANGLO	
F7F5 FF 70B8	STX	MEMADR	
	*GET BYTE STORED AT TEST LOCATION & SAVE IT		
F7F8 A6 00	TEST1	LDA A X	
F7FA 36		PSH A	
F7FB 6F 00	CLR	X	ZERO THE LOCATION
F7FD 6D 00	TST	X	TEST IT
F7FF 27 05	BEQ	TEST2	OK IF = ZERO
 	*CAN'T CLEAR LOCATION		
F801 CE FF32	LDX	#MSGCL	
F804 20 1E	BRA	TEST4	
F806 6A 00	TEST2	DEC X	SET LOCATION TO FF
F808 86 FF	LDA A	#\$FF	
F80A A1 00	CMP A	X	DID IT GET SET TO FF?
F80C 27 05	BEQ	TEST3	
 	*CAN'T SET LOCATION TO ONE'S		
F80E CE FF3D	LDX	#MSGCSO	
F811 20 11	BRA	TEST4	
F813 FE 70B8	TEST3	LDX MEMADR	GET LOCATION BEING TESTED
F816 32	PUL A		
F817 A7 00	STA A X		RESTORE PREVIOUS CONTENT
 	*HIT END OF TEST RANGE?		
F819 BC 7017	CPX	RANGHI	
F81C 27 CC	BEQ	SEAR10	YES, ALL DONE
 	*NO, MOVE TO TEST NEXT LOCATION		
F81E 08	INX		
F81F FF 70B8	STX	MEMADR	
F822 20 D4	BRA	TEST1	
 	**LOCATION IS BAD		
F824 FF 70BC	TEST4	STX TEMP3	SAVE ERROR MESSAGE TEMPORARILY
F827 CE 70B8	LDX	#MEMADR	
F82A BD FC04	JSR	OUT2BY	TYPE OUT BAD ADDRESS,
F82D BD FBF7	JSR	OUTEQ	AN EQUAL SIGN,
F830 FE 70B8	LDX	MEMADR	
F833 BD FBFD	JSR	OUT1BY	ITS CONTENT,
F836 BD FBF1	JSR	OUTSP	A SPACE,

F839 FE 70BC	LDX	TEMP3	
F83C BD FE4B	JSR	OUTSTR	AND THE TYPE OF ERROR
F83F BD FEC7	JSR	DOCRLF	SEND CR-LF
F842 20 CF	BRA	TEST3	

* INT - SET UP INTERRUPT POINTER			
F844 BD FB3C INT	JSR	NUMINX	GET POINTER IN IX
F847 FF 7000	STX	INTVEC	SAVE IT
F84A 20 2C	BRA	COMPAL	

*NMI - SET UP NON-MASKABLE INTERRUPT POINTER			
F84C BD FB3C NMI	JSR	NUMINX	GET POINTER IN IX
F84F FF 7002	STX	NMIVEC	SAVE IT
F852 20 24	BRA	COMPAL	

*SWI - SET UP SWI POINTER			
F854 BD FB3C SWI	JSR	NUMINX	GET POINTER IN IX
F857 FF 7004	STX	SWIVEC	SAVE IT
F85A 20 1C	BRA	COMPAL	

*COMPARE - OUTPUT SUM & DIFFERENCE OF TWO INPUT NUMBERS			
F85C BD FB3C COMPAR	JSR	NUMINX	GET FIRST NUMBER
F85F FF 7015	STX	RANGLO	PUT IT IN RANGLO
F862 BD FB3C	JSR	NUMINX	GET SECOND NUMBER
F865 FF 7013	STX	NBRHI	SAVE IT IN NBRHI
*COMPUTE AND OUTPUT THE SUM			
F868 BD FAD6	JSR	SUMNUM	COMPUTE SUM
F86B CE FF4E	LDX	#MSG1S	GET ITS TITLE
F86E 8D 0B	BSR	OUTSD	OUTPUT TITLE & SUM
F870 BD FAE9	JSR	DIFNUM	COMPUTE DIFFERENCE
F873 CE FF56	LDX	#MSGDIS	GET ITS TITLE
F876 8D 03	BSR	OUTSD	OUTPUT TITLE & DIFFERENCE
F878 7E F46F COMPAL	JMP	NOMORE	
*COMPUTE AND OUTPUT THE RESULT			
F87B BD FE4B OUTSD	JSR	OUTSTR	OUTPUT IT
F87E CE 7017	LDX	#RANGHI	GET RESULT
F881 BD FC04	JSR	OUT2BY	DISPLAY RESULT
F884 39	RTS		

*DUMP - DUMP PORTION OF MEMORY, IN MIKBUG FORMAT, TO			
* A SPECIFIED ACIA ADDRESS			

*GET ADDRESS RANGE: START IN RANGLO (2 BYTES), END IN
 * RANGHI (2 BYTES)
 *IF NO ADDRESS RANGE IS GIVEN, USE WHATEVER IS IN
 * RANGLO & RANGHI

F885 BD FAFC DUMP JSR GTRANG

F888 7F 70C0 CLR TEMP5 INITIALIZE TO DUMP TO TERMINAL

*LOOK FOR A "TO" MODIFIER

F88B 86 02	DUMP1	LDA A #2	
F88D BD F9C7		JSR COMAND	
F890 27 13		BEQ DUMP4	
F892 2F 7C	DUMP2	BLE DUMP10	ERROR IF BAD SYNTAX
F894 81 01		CMP A #1	TO?
F896 27 02		BEQ DUMP3	
F898 20 F1		BRA DUMP1	GO LOOK FOR ANOTHER MODIFIER

F89A BD FB3C	DUMP3	JSR NUMINX	GET "TO" ADDRESS
F89D FF 7027		STX OUTADR	SAVE IT
F8A0 7C 70C0		INC TEMP5	REMEMBER THIS
F8A3 20 E6		BRA DUMP1	GO LOOK FOR ANOTHER MODIFIER

F8A5 7D 70C0	DUMP4	TST TEMP5	
F8A8 27 03		BEQ DUMP5	
F8AA 7C 7026		INC OUTFLG	SET FLAG FOR PROPER OUTPUT
	*		DEVICE
F8AD 8D 64	DUMP5	BSR NULLS	SEND SOME NULLS

*MIKBUG MODE

*OUTPUT AN "S0" TYPE RECORD

F8AF CE FF60		LDX #MSG0	
F8B2 BD FE4B		JSR OUTSTR	

*COMPUTE # OF BYTES TO OUTPUT (RANGE END - RANGE START
 * + 1)

*SUBTRACT LO BYTES

F8B5 B6 7018	DUMP6	LDA A RANGHI+1	
F8B8 B0 7016		SUB A RANGLO+1	

*SUBTRACT HI BYTES

F8BB F6 7017		LDA B RANGHI	
F8BE F2 7015		SBC B RANGLO	

*NON-ZERO HI BYTE IMPLIES LOTS TO OUTPUT

F8C1 26 04		BNE DUMP7	
------------	--	-----------	--

*HI BYTE DIFF IS ZERO

F8C3 81 10		CMP A #16	LO BYTE OF DIFF 0 TO 15
F8C5 25 02		BCS DUMP8	IF YES, TO DUMP8
F8C7 86 0F	DUMP7	LDA A #15	NO, LO BYTE IS 16-255: SET

* BYTES TO 15

*TO GET FRAME COUNT, ADD 1 (DIFF OF 0 IMPLIES 1
 * OUTPUT) + # OF DATA BYTES,

* + 2 ADDR BYTES + 1 CHECKSUM BYTE

F8C9 8B 04		DUMP8 ADD A #4	
------------	--	----------------	--

F8CB B7 70BC	STA A TEMP3	TEMP3 IS THE FRAME COUNT
F8CE 80 03	SUB A #3	
F8D0 B7 70BE	STA A TEMP4	TEMP4 IS THE RECORD BYTE COUNT
*OUTPUT A MIKBUG "S1"		
F8D3 CE FF74	LDX #MSG\$1	
F8D6 BD FE4B	JSR OUTSTR	HEADER DATA RECORD
F8D9 5F	CLR B	ZERO CHECKSUM
*PUNCH FRAME COUNT		
F8DA CE 70BC	LDX #TEMP3	
F8DD 8D 3E	BSR OUTP2	

*PUNCH ADDRESS

F8DF CE 7015	LDX #RANGLO	
F8E2 8D 39	BSR OUTP2	
F8E4 8D 37	BSR OUTP2	

*OUTPUT DATA

F8E6 FE 7015	LDX RANGLO	
F8E9 8D 32 DUMP9	BSR OUTP2	OUTPUT DATA BYTE
F8EB 7A 70BE	DEC TEMP4	DEC BYTE COUNT
F8EE 26 F9	BNE DUMP9	

*COMPLEMENT AND PUNCH THE CHECKSUM

F8F0 FF 7015	STX RANGLO	SAVE MEMORY POINTER
F8F3 53	COM B	COMPLEMENT CHECKSUM
F8F4 37	PSH B	PUT IT ON STACK
F8F5 30	TSX	LET IX POINT TO IT
F8F6 8D 25	BSR OUTP2	OUTPUT CHECKSUM
F8F8 33	PUL B	PULL IT OFF STACK
F8F9 FE 7015	LDX RANGLO	RESTORE MEMORY POINTER
F8FC 09	DEX	
F8FD BC 7017	CPX RANGHI	HIT END OF RANGE?
F900 26 B3	BNE DUMP6	

*YES, OUTPUT AN "S9" RECORD

F902 CE FF7B	LDX #MSG\$9	
F905 BD FE4B	JSR OUTSTR	
F908 8D 09	BSR NULLS	GENERATE BLANK TAPE
F90A 7F 7026	CLR OUTFLG	SET TO TERMINAL OUTPUT
F90D 7E F46F	JMP NOMORE	ALL DONE
F910 7E F457 DUMP10	JMP BADSYN	BAD SYNTAX

*SEND A STRING OF NULLS

F913 C6 1E	NULLS LDA B #30	
F915 4F	CLR A	
F916 BD FE76	NULLS1 JSR OUTCHR	
F919 5A	DEC B	
F91A 26 FA	BNE NULLS1	
F91C 39	RTS	

*OUTPUT A BYTE POINTED TO BY IX AS 2 HEX CHARACTERS

F91D EB 00	OUTP2 ADD B X	UPDATE CHECKSUM
F91F BD FBFD	JSR OUT1BY	

F922 08		INX	
F923 39		RTS	

	*LOAD - LOAD A MIKBUG TAPE		
	*LOOK FOR A "FROM" MODIFIER		
F924 86 07	LOAD	LDA A #7	IN LIST 7
F926 BD F9C7		JSR COMAND	
F929 2B E5		BMI DUMP10	ERROR, UNRECOGNIZABLE MODIFIER
F92B 27 09		BEQ LOAD1	
F92D BD FB3C		JSR NUMINX	GET "FROM" ADDRESS
F930 FF 7024		STX INPADR	SAVE IT
F933 7C 7023		INC INPFLG	SET FLAG FOR NON-TERMINAL ACIA
	*KEEP READING CHARACTERS UNTIL AN "S" IS READ		
F936 BD FE59	LOADI	JSR INPCHR	GET A CHAR
F939 81 53		CMP A #'S	IS IT AN S?
F93B 26 F9		BNE LOAD1	
	*GOT AN "S", EXAMINE NEXT CHARACTER		
F93D BD FE59		JSR INPCHR	
F940 81 39		CMP A #'9	DONE IF ITS A "9"
F942 27 2E		BEQ LOAD4	
F944 81 31		CMP A #'1	IS IT A "1"?
F946 26 EE		BNE LOAD1	IF NOT, LOOK FOR NEXT "S"
*VALID S1 RECORD			
F948 7F 70E1		CLR CKSM	CLEAR CHECKSUM
*READ RECORD BYTE COUNT			
F94B BD F986		JSR RDBYTE	
F94E 80 02		SUB A #2	
F950 B7 70E0		STA A BYTECT	SAVE COUNT MINUS 2 ADDRESS BYTES
F953 8D 23		BSR BLDADR	BUILD ADDRESS
F955 8D 2F	LOAD2	BSR RDBYTE	READ A DATA BYTE INTO ACCA
F957 7A 70E0		DEC BYTECT	COUNT IT
F95A 27 05		BEQ LOAD3	IF DONE WITH RECORD, CHECK CHECKSUM
*			
F95C A7 00		STA A X	NOT DONE, STORE BYTE IN MEMORY
F95E 08		INX	ON TO NEXT MEMORY ADDRESS
F95F 20 F4		BRA LOAD2	
	*RECORD READ IN COMPLETE		
F961 7C 70E1	LOAD3	INC CKSM	TEST CHECKSUM BY ADDING 1
F964 27 D0		BEQ LOAD1	IF OK, RESULT SHOULD BE ZERO
	*RECORD CHECKSUM ERROR		
F966 CE FF22		LDX #MSGNVE	SAY SO
F969 BD FE4B		JSR OUTSTR	

F96C CE 70B8	LDX #TEMP1	GET RECORD ADDRESS OF IT
F96F BD FC04	JSR OUT2BY	TYPE IT TOO
F972 7F 7023	LOAD4 CLR	RESET FLAG TO NORMAL TERMINAL
	*	INPUT
F975 7E F46F	JMP NOMORE	

* BUILD ADDRESS

F978 8D 0C	BLDADR BSR RDBYTE	
F97A B7 70B8	STA A TEMP1	
F97D 8D 07	BSR RDBYTE	
F97F B7 70B9	STA A TEMP1+1	
F982 FE 70B8	LDX TEMP1	
F985 39	RTS	
F986 8D 10	RDBYTE BSR INHEX	GET LEFT HEX DIGIT
	*MOVE TO HI 4 BITS	
F988 48	ASL A	
F989 48	ASL A	
F98A 48	ASL A	
F98B 48	ASL A	
F98C 16	TAB	SAVE IT IN ACCA
F98D 8D 09	BSR INHEX	GET RIGHT HEX DIGIT
F98F 1B	ABA	COMBINE THEM IN ACCA
	*UPDATE THE CHECKSUM	
F990 16	TAB	
F991 FB 70E1	ADD B CKSM	
F994 F7 70E1	STA B CKSM	
F997 39	RTS	

* INPUT A HEX CHAR & CONVERT TO INTERNAL FORM

F998 BD FE59	INHEX JSR INPCHR	INPUT A CHAR
F99B 80 30	SUB A #\$30	
F99D 2B 0F	BMI INHEX2	NOT HEX IF BELOW ASCII "I"
F99F 81 09	CMP A #\$09	
F9A1 2F 0A	BLE INHEX1	OK IF ASCII "9" OR LESS
F9A3 81 11	CMP A #\$11	BELLOW ASCII "A"?
F9A5 2B 07	BMI INHEX2	ERROR IF IT IS
F9A7 81 16	CMP A #\$16	OVER ASCII "F"?
F9A9 2E 03	BGT INHEX2	ERROR IF IT IS
F9AB 80 07	SUB A #7	CONV ASCII A-F TO HEX A-F
F9AD 39	INHEX1 RTS	
	*ERROR - CHAR NOT HEX, SAY SO	
F9AE CE FF8B	INHEX2 LDX #MSGCNH	
F9B1 BD FE4B	JSR OUTSTR	
F9B4 39	RTS	

*DELAY - DELAY SPECIFIED # OF MILLISECONDS

F9B5 BD FB3C	DELAY JSR NUMINX	GET DELAY TIME
F9B8 8D 03	BSR TIMDEL	
F9BA 7E F46F	JMP NOMORE	

*TIME DELAY SUBROUTINE

*IX IS INPUT AS THE # OF MILLISECONDS TO DELAY

```

*ACCA IS ALTERED
*ACCB IS PRESERVED
*ADJ TIMCON SO (6*TIMCON*CYCLE TIME=1 MS)
F9BD B6 70DE TIMDEL LDA A TIMCON
*ENTER A 6 CYCLE LOOP
F9C0 4A      TIMDE1 DEC A
F9C1 26 FD    BNE     TIMDE1

F9C3 09      DEX          DECREMENT MILLISECOND COUNTER
F9C4 26 F7    BNE     TIMDEL
F9C6 39      RTS

```

*=====

* C O M M A N D L I S T S C A N N I N G R
* O U T I N E

*THIS ROUTINE SEEKS A MATCH OF THE CHARACTERS POINTED
* AT
*BY THE INPUT LINE SCANNING POINTER TO ONE OF THE
* COMMANDS
*IN A LIST SPECIFIED BY ACCA.
*THE RESULT OF THE SCAN FOR A MATCH IS RETURNED IN
* ACCA,
* AS FOLLOWS:
*
* ACCA=-1: THE MATCH WAS UNSUCCESSFUL. THE SYNTAX
* POINTER (SYNPTR) WAS NOT UPDATED
* (ADVANCED).
*
* ACCA= 0: THE MATCH WAS UNSUCCESSFUL SINCE THERE
* WERE
* NO MORE CHARACTERS, I.E., THE END OF
* THE
* LINE WAS REACHED.
*
* ACCA=+N: SUCCESSFUL MATCH. THE SYNTAX POINTER
* WAS UPDATED
* TO THE FIRST CHARACTER FOLLOWING THE
* COMMAND
* DELIMITER. ACCA HOLDS THE NUMBER OF
* THE
* COMMAND MATCHED.
*GLOBAL VARIABLES FOR EXTERNAL COMMUNICATION
*SYNPTR - GOOD SYNTAX INPUT LINE CHAR POINTER
*LINPTR - INPUT LINE CHARACTER POINTER
*DELIM - CLASS OF PERMISSIBLE COMMAND DELIMITERS

*TEMPORARY 2 BYTE INTERNAL VARIABLES
*LISPTR - COMMAND LIST CHARACTER POINTER

*TEMPORARY 1 BYTE INTERNAL VARIABLES

*NUMMAT - NUMBER OF CHARACTERS THAT SUCCESSFULLY MATCH
*LISNUM - # OF LIST WITHIN WHICH A MATCH WILL BE SOUGHT
*COMNUM - COMMAND NUMBER MATCHED

*CONSTANTS USED
*CR - CARRIAGE RETURN
*LF - LINE FEED

*ACCB & IX ARE NOT PRESERVED

F9C7 B7 70D5 COMAND STA A LISNUM SAVE LIST # TO MATCH WITHIN
*TEST IF WE ARE AT THE END OF THE LINE
F9CA BD FA69 JSR SKPDLM
F9CD 24 02 BCC INILST
F9CF 4F CLR A
F9D0 39 RTS

*INITIALIZE THE COMMAND LIST POINTER TO ONE LESS THAN
* THE BEGINNING OF THE COMMAND LI
STS

F9D1 FE 7008 INILST LDX COMADR ENTRY POINT

*MOVE TO THE BEGINNING OF THE DESIRED COMMAND LIST

F9D4 B6 70D5 LDA A LISNUM SEARCH FOR "STRING" # LISNUM
F9D7 C6 0A LDA B #LF USE LF AS A "STRING" TERMINATOR
F9D9 8D 76 BSR FNDSTR
F9DB FF 70D7 STX LISPTR

*THE LIST POINTER, LISPTR, NOW POINTS TO ONE LESS THAN
* THE FIRST CHARACTER
*OF THE FIRST COMMAND IN THE DESIRED LIST
* INITIALIZE THE COMMAND # TO 1

F9DE 86 01 LDA A #1
F9E0 B7 70D6 STA A COMNUM

*RESET INPUT LINE POINTER TO: 1) BEGINNING OF LINE, OR
* TO
* 2) POINT WHERE LAST SUCCESSFUL SCAN TERMINATED

F9E3 FE 700A CMD3 LDX SYNPTR
F9E6 FF 700C STX LINPTR

F9E9 7F 70D4 CLR NUMMAT CLEAR NUMBER OF CHARACTERS
* MATCHED
F9EC BD FCC0 CMD4 JSR GETCHR GET INPUT LINE CHAR IN ACCB
F9EF BD FA94 JSR TSTDLM TEST FOR A DELIMITER
F9F2 26 13 BNE MATCH SUCCESS (FOUND DELIMITER) IF
* NOT = ZERO

F9F4 BD FCCD JSR GETLST GET COMMAND LIST CHAR IN ACCA

F9F7 81 0A	*	CMP A #LF	HAS END OF COMMAND LIST BEEN REACHED?
F9F9 27 16		BEQ NMATCH	IF SO, POTENTIAL MATCH FAILURE
F9FB 81 0D		CMP A #CR	HAS END OF COMMAND BEEN REACHED?
F9FD 27 12		BEQ NMATCH	IF SO, POTENTIAL MATCH FAILURE
F9FF 11		CBA	COMPARE THE TWO CHARACTERS
FA00 26 19	*	BNE NEXCOM	MATCH NOT POSSIBLE ON THIS COMMAND
FA02 7C 70D4	*THEY MATCH, COMPARE THE SUCCEEDING CHARACTERS	INC NUMMAT	INC NUMBER OF CHARACTERS MATCHED
FA05 20 E5		BRA CMD4	

*SUCCESSFUL MATCH - RETURN COMMAND NUMBER MATCHED IN ACCA			
*			
FA07 B6 70D6	MATCH	LDA A COMNUM	
FA0A FE 700C		LDX LINPTR	
FA0D FF 700A		STX SYNPTR	UPDATE GOOD SYNTAX POINTER
FA10 39		RTS	

*NO MATCH			
*DID AT LEAST ONE MATCH?			
FA11 7D 70D4	NMATCH	TST NUMMAT	
FA14 27 05		BEQ NEXCOM	TO NEXT COMMAND IF NONE MATCHED
*AT LEAST ONE MATCHED - TEST FOR DELIMITER (NON-MATCHING CHAR)			
*			
FA16 BD FA94		JSR TSTDLM	
FA19 26 EC		BNE MATCH	IF A DELIMITER, MATCH HAS BEEN ACHIEVED
*			
*ILLEGAL DELIMITER			
*MOVE TO NEXT COMMAND WITHIN LIST			
FA1B BD FCCD	NEXCOM	JSR GETLST	GET NEXT COMMAND LIST CHARACTER
FA1E 81 0A		CMP A #LF	END OF THIS LIST?
FA20 27 09		BEQ MFAIL	IF SO, NOTHING ON LIST MATCHED
FA22 81 0D		CMP A #CR	IS IT A CR?
FA24 26 F5		BNE NEXCOM	IF NOT, MOVE TO NEXT CHARACTER
FA26 7C 70D6		INC COMNUM	YES, INC COMMAND NUMBER
FA29 20 B8		BRA CMD3	

*MATCH FAILURE - NO MATCH POSSIBLE WITHIN THIS LIST			
FA2B 4F	MFAIL	CLR A	
FA2C 4A		DEC A	
FA2D 39		RTS	

```

=====
*THIS ROUTINE TYPES OUT COMMAND NUMBER "COMNUM"
*THE LIST IS SPECIFIED IN ACCA
*ACCB & IX ARE PRESERVED

```

FA2E FF 70CE	TYPCMD	STX XTEMP	
FA31 37		PSH B	
FA32 CE FCD6		LDX #COMLST-1	MOVE TO HEAD OF COMMAND
	*		LISTS
FA35 C6 0A	LDA B	#LF	AND LIST TERMINATOR
FA37 8D 18	BSR	FNDSTR	GO TO HEAD OF DESIRED LIST
FA39 B6 70D6	LDA A	COMNUM	GET COMMAND NUMBER
FA3C C6 0D	LDA B	#CR	GET COMMAND TERMINATOR
FA3E 8D 11	BSR	FNDSTR	GO TO HEAD OF DESIRED COMMAND

FA40 08	TYPCM1	INX	MOVE TO NEXT CHARACTER
FA41 A6 00	LDA A	X	GET A COMMAND CHARACTER
FA43 81 0D	CMP A	#CR	IS IT A COMMAND TERMINATOR?
FA45 27 05	BEQ	TYPCM2	IF SO, RETURN
FA47 BD FE76	JSR	OUTCHR	NO, TYPE IT
FA4A 20 F4	BRA	TYPCM1	

FA4C FE 70CE	TYPCM2	LDX XTEMP	
FA4F 33		PUL B	
FA50 39		RTS	

```

=====
*MOVE TO BEGINNING OF DESIRED STRING NUMBER (IN ACCA)
*EACH STRING IS TERMINATED BY AN END OF STRING
*CHARACTER (IN ACCB)
*THE INDEX REGISTER IS ASSUMED INITIALIZED POINTING TO
*ONE LESS THAN THE FIRST CHARACTER OF THE FIRST STRING
*ACCA, ACCB & IX ARE NOT PRESERVED
*LOCAL VARIABLES
*STRNUM - STRING # TO FIND
*EOSCHR - "END OF STRING" CHARACTER

```

FA51 B7 70BA	FNDSTR	STA A STRNUM	SAVE STRING NUMBER
FA54 F7 70BB		STA B EOSCHR	SAVE TERMINATOR
FA57 5F		CLR B	
FA58 5C	FNDST1	INC B	STRING 1 IS THE FIRST STRING
FA59 F1 70BA		CMP B STRNUM	IS THIS THE RIGHT STRING?
FA5C 27 0A		BEQ FNDST3	IF SO, DONE

		*NO, SWALLOW UP CHARACTERS UNTIL AN END OF STRING CHAR	
		*	IS HIT
FA5E 08	FNDST2	INX	BUMP POINTER TO NEXT ONE
FA5F A6 00		LDA A X	GET CHAR POINTED AT
FA61 B1 70BB		CMP A EOSCHR	END OF STRING HIT?
FA64 27 F2		BEQ FNDST1	IF IT IS, BUMP THE STRING
	*		COUNTER
FA66 20 F6		BRA FNDST2	NO, MOVE ON TO NEXT CHAR

FA68 39

FNDST3 RTS

IX SET PROPERLY, RETURN

```

*=====
*SKIP LEADING DELIMITERS
*THIS ROUTINE SHOULD BE CALLED PRIOR TO SCANNING FOR
*      ANY INFORMATION
*ON THE INPUT LINE
*THE CURRENT CHARACTER IS IGNORED IF THE SCANNING
*      POINTER IS AT THE
*BEGINNING OF A LINE.  IF NOT, THE SCANNING POINTER
*      SKIPS OVER SPACES
*AND COMMAS UNTIL AN END OF LINE OR NON-DELIMITER IS
*      FOUND.
*THE CARRY BIT IS SET IF AN END OF LINE IS ENCOUNTERED.

*ACCA, ACCB, & IX ARE NOT PRESERVED

```

FA69 0C SKPDLM CLC
 FA6A 7D 700E TST BOLFLG AT BEGINNING OF LINE?
 FA6D 2E 0B BGT SKPDL2

```

*LOOK AT CURRENT INPUT CHARACTER
FA6F FE 700A SKPDL1 LDX  SYNPTR  GET POINTER TO IT
FA72 A6 00       LDA A  X       GET CHAR
FA74 8D 13       BSR    TSTEOL  TEST FOR END OF LINE
FA76 26 02       BNE    SKPDL2
FA78 0D         SEC     YES, END HIT, SET CARRY
FA79 39         RTS
* "PEEK" AT NEXT CHAR IN LINE
FA7A E6 01       SKPDL2 LDA B  1,X   GET IT
FA7C 8D 16       BSR    TSTDLM  SEE IF ITS A DELIMITER
FA7E 26 01       BNE    SKPDL3
FA80 39         RTS     ITS NOT, RETURN

```

```

*NEXT CHAR IS A DELIMITER
FA81 BD FCC0 SKPDL3 JSR  GETCHR  MOVE TO NEXT CHAR IN INPUT LINE
FA84 FF 700A     STX    SYNPTR  UPDATE SYNTAX POINTER
FA87 20 E6     BRA    SKPDL1  GO TEST FOR END OF LINE

```

```

*=====
*TEST FOR END-OF-LINE CHARACTER
*Z BIT OF CC REG SET IF CHAR IN ACCA IS A TERMINATOR
*ACCA, ACCB, & IX ARE PRESERVED
FA89 81 0D     TSTEOL CMP A  #CR     CARRIAGE RETURN?
FA8B 27 06       BEQ    TSTE01
FA8D 81 0A       CMP A  #LF     LINE FEED? (CONTINUED LINES)
FA8F 27 02       BEQ    TSTE01
FA91 81 3B       CMP A  #';     FOR SEVERAL COMMANDS ON ONE LINE
FA93 39         TSTE01 RTS

```

```

*=====
*CHECK THE CHARACTER IN ACCB
*AGAINST THE DELIMITER(S) SPECIFIED BY VARIABLE DELIM
*ACCB & IX ARE PRESERVED
*ACCA IS SET TO 0 IF ACCB IS NOT A DELIMITER, TO 1 IF
*      IT IS
*      IF DELIM=1, SPACE IS DELIMITER
*      IF DELIM=2, COMMA IS DELIMITER
*      IF DELIM=3, SPACE OR COMMA IS DELIMITER
*      IF DELIM=4, ANY NON-ALPHANUMERIC IS A DELIMITER
*TEST FOR END-OF-LINE (LOGICAL OR PHYSICAL)

```

FA94 37	TSTDLM PSH B
FA95 17	TBA
FA96 8D F1	BSR TSTEOL
FA98 33	PUL B
FA99 27 35	BEQ DLMYES

FA9B B6 700F	LDA A DELIM
FA9E 81 01	CMP A #1
FAA0 26 06	BNE ISDLM2
FAA2 C1 20	CMP B #32 WANT A SPACE - IS IT?
FAA4 26 2D	BNE DLMNO
FAA6 20 28	BRA DLMYES

FAA8 81 02	ISDLM2 CMP A #2
FAAA 26 06	BNE ISDLM3
FAAC C1 2C	TSTCMA CMP B #', WANT A COMMA - IS IT?
FAAE 26 23	BNE DLMNO
FABA 20 1E	BRA DLMYES
FAB2 81 03	ISDLM3 CMP A #3
FAB4 26 06	BNE ISDLM4
FAB6 C1 20	CMP B #32 WANT EITHER, IS IT A SPACE?
FAB8 27 16	BEQ DLMYES
FABA 20 F0	BRA TSTCMA OR A COMMA?

FABC 81 04	ISDLM4 CMP A #4
FABE 26 15	BNE ERROR ERROR IF DELIM NOT 1-4
*TEST IF CHAR IS 0 TO 9 INCLUSIVE	
FAC0 C1 30	CMP B #'0
FAC2 2D 0C	BLT DLMYES
FAC4 C1 39	CMP B #'9
FAC6 2F 0B	BLE DLMNO

*TEST IF CHAR IS A TO 9 INCLUSIVE	
FAC8 C1 41	CMP B #'A
FACA 2D 04	BLT DLMYES
FACC C1 5A	CMP B #'Z
FACE 2F 03	BLE DLMNO
*OVER Z - ITS A DELIMITER	

*CHAR IN ACCB IS A DELIMITER	
FAD0 86 01	DLMYES LDA A #1
FAD2 39	RTS

*CHAR IN ACCB IS NOT A DELIMITER

FAD3 4F DLMNO CLR A

FAD4 39 RTS

* ERROR IN SPECIFYING DELIMITER CLASS

FAD5 3F ERROR SWI HAVE MONITOR TYPE OUT PERTINENT
* STATISTICS

*=====

*ADD THE 2 BYTE NUMBER STORED IN (RANGLO,RANGLO+1) TO
* THE NUMBER

* STORED IN (NBRHI,NBRLO) AND PUT THE RESULT IN
* (RANGHI,RANGHI+1)

*ACCB & IX ARE PRESERVED

*ACCA IS ALTERED

*ADD LO ORDER BYTES

FAD6 B6 7016 SUMNUM LDA A RANGLO+1

FAD9 BB 7014 ADD A NBRLO

FADC B7 7018 STA A RANGHI+1

*ADD HI ORDER BYTES

FADF B6 7015 LDA A RANGLO

FAE2 B9 7013 ADC A NBRHI

FAE5 B7 7017 STA A RANGHI

FAE8 39 RTS

*=====

*SUBTRACT THE 2 BYTE NUMBER STORED IN (NBRHI,NBRLO)

* FROM THE

*TWO BYTE NUMBER STORED IN (RANGLO,RANGLO+1) AND PUT
* THE

*RESULT IN (RANGHI,RANGHI+1)

*ACCB & IX ARE PRESERVED

*ACCA IS ALTERED

*SUBTRACT LO ORDER BYTES

FAE9 B6 7016 DIFNUM LDA A RANGLO+1

FAEC B0 7014 SUB A NBRLO

FAEF B7 7018 STA A RANGHI+1

*=====

*SUBTRACT HI ORDER BYTES

FAF2 B6 7015 LDA A RANGLO

FAF5 B2 7013 SBC A NBRHI

FAF8 B7 7017 STA A RANGHI

FAFB 39 RTS

```

=====
* THIS ROUTINE SCANS THE INPUT LINE FOR A PAIR OF
* NUMBERS
* REPRESENTING AN ADDRESS RANGE. A COLON SEPARATING THE
* PAIR IMPLIES "THRU", WHILE AN "!" IMPLIES "THRU THE
* FOLLOWING"
* E.G., 100:105 IS EQUIVALENT TO 100!5
* A SINGLE NUMBER IMPLIES A RANGE OF 1
*
* ON RETURN (RANGLO,RANGLO+1) HOLDS THE RANGE START, AND
* (RANGHI,RANGHI+1) HOLDS THE RANGE END
* ACCA, ACCB, & IX ARE NOT PRESERVED

```

FAFC 8D 49	GTRANG	BSR	NUMBER	PICK UP FIRST NUMBER
FAFE 2E 03		BGT	GTRAN1	
FB00 2D 09		BLT	GTRAN2	
FB02 39		RTS		NOTHING MORE ON INPUT LINE

*GOOD SINGLE NUMBER - TRANSFER IT TO RANGLO				
FB03 FE 7013	GTRAN1	LDX	NBRHI	
FB06 FF 7015		STX	RANGLO	
FB09 20 0D		BRA	GTRAN3	AND TO RANGHI
*BAD NUMBER, BUT IS IT BAD DUE TO A ":" OR "!"				
* DELIMITER?				
*GET THE TERMINATOR FOR THE FIRST NUMBER				
FB0B FE 700C	GTRAN2	LDX	LINPTR	
FB0E A6 00		LDA A	X	
FB10 81 3A		CMP A	#':	WAS IT A COLON?
FB12 26 0C		BNE	GTRAN4	IF NOT, GO TEST FOR "!"
FB14 8D 1A		BSR	GTRAN8	WAS ":" , PROCESS FIRST NUMBER &
*				GET NEXT ONE
FB16 2F 0E		BLE	GTRANS	ILLEGAL IF END OF LINE OR
*				NON-NUMERIC

*TRANSFER SECOND NUMBER TO RANGHI				
FB18 FE 7013	GTRAN3	LDX	NBRHI	
FB1B FF 7017		STX	RANGHI	
FB1E 20 0D		BRA	GTRAN7	
FB20 81 21	GTRAN4	CMP A	#'!	WAS DELIMITER A "!"?
FB22 27 03		BEQ	GTRAN6	IF YES, GET 2ND NUMBER

* ILLEGAL DELIMITER, RETURN				
FB24 4F		CLR A		
FB25 4A		DEC A		
FB26 39		GTRAN5	RTS	
FB27 8D 07	GTRAN6	BSR	GTRAN8	WAS "!" , PROCESS FIRST NUMBER &
*				GET NEXT ONE
FB29 2F FB		BLE	GTRANS	

FB2B 8D A9 **BSR** **SUMNUM** **COMPUTE RANGE END, PUT INTO RANGHI**

*SUCCESSFUL EXIT
FB2D 86 01 GTRAN7 LDA A #1
FB2F 39 RTS

```

*UPDATE SYNTAX POINTER, MOVE FIRST NUMBER TO RANGLO, &
*
*                                         GET 2ND NUMBER
GTRAN8 STX      SYNPTR   UPDATE SYNTAX POINTER
          LDX      NBRHI    GET FIRST NUMBER OF THE PAIR
          STX      RANGLO   SAVE IT IN "LOW RANGE" VALUE
          BSR      NUMBER   PICK UP THE SECOND NUMBER OF
*                                         THE PAIR

```

*===== *GET A 2 BYTE NUMBER & RETURN IT IN THE INDEX REGISTER

```
FB3C 8D 09    NUMINX BSR      NUMBER
FB3E 2E 03          BGT      NUMINI
FB40 7E F457        JMP      BADSYN
FB43 FE 7013 NUMINI LDX      NBRHI
FB46 39          RTS
```

```
* ======  
* SCAN FOR A NUMBER  
* RETURN THE MOST SIGNIFICANT BYTE IN NBRHI  
* AND THE LEAST SIGNIFICANT BYTE IN NBRLO  
* THE RESULT OF THE SCAN FOR A NUMBER IS RETURNED IN  
* ACCA AS FOLLOWS:  
* ======
```

*
* ACCA=-1: THE MATCH WAS UNSUCCESSFUL. THE SYNTAX
* POINTER (SYNPTB) WAS NOT UPDATED.
*

* ACCA= 0: THE SCAN WAS UNSUCCESSFUL SINCE THERE
* WERE NO
* MORE CHARACTERS. (I.E., THE END OF THE
* LINE WAS ENCOUNTERED.)

* ACCA=+1: THE SCAN WAS SUCCESSFUL. THE SYNTAX
* POINTER
* WAS UPDATED TO THE FIRST CHARACTER
* FOLLOWING
* THE COMMAND.

```
*IX IS PRESERVED  
*GLOBAL VARIABLES FOR EXTERNAL COMMUNICATION  
*NBRHI - NUMBER HI BYTE  
*NBRLO - NUMBER LO BYTE  
*IBCODE - INPUT BASE CODE  
*DBCODE - DISPLAY BASE CODE
```

```
* LOCAL VARIABLES
* NBR2X - USED IN DECIMAL CONVERSION
* XTEMP2 - SAVES IX
```

```

        * INITIALIZE BOTH BYTES TO ZERO
FB47 FF 70D2 NUMBER STX      XTEMP2    SAVE IX
FB4A 7F 7013           CLR      NBRH1
FB4D 7F 7014           CLR      NBRLO

        * INITIALIZE THE LINE SCANNING POINTER
FB50 FE 700A           LDX      SYNPTR
FB53 FF 700C           STX      LINPTR

        * ARE WE AT END OF LINE?
FB56 BD FA69           JSR      SKPDLM
FB59 24 05           BCC      NUMLUP
FB5B FE 70D2           LDX      XTEMP2
FB5E 4F           CLR A      YES, ZERO ACCA
FB5F 39           RTS

FB60 BD FCC0 NUMLUP JSR      GETCHR   GET A CHARACTER FROM THE INPUT
*                                         LINE INTO ACCB

        * TEST FOR A DELIMITER
FB63 BD FA94           JSR      TSTDLM
FB66 26 65           BNE      GUDNUM  GOOD DELIMITER IF ACCA NON-ZERO

        * NOT A DELIMITER, TEST IF CHARACTER IS < ASCII 0
FB68 C0 30           SUB B      #'0      SUBTRACT ASCII 0
FB6A 2B 6D           BMI      CONERR  ERROR IF LESS

        * DETERMINE INPUT BASE & GO TO RIGHT ROUTINE
FB6C B6 7010           LDA A      IBCODE
FB6F 81 01           CMP A      #1
FB71 27 08           BEQ      HEXNUM

FB73 81 02           CMP A      #2
FB75 27 1E           BEQ      DECNUM

FB77 81 03           CMP A      #3
FB79 27 41           BEQ      OCTNUM

        * DEFAULT AN ILLEGAL INPUT BASE TO HEX
*****
        * INPUT A HEX NUMBER
        * TEST FOR A LEGAL DIGIT
FB7B C1 09           HEXNUM CMP B      #$09
FB7D 2F 0A           BLE      HEXN1    OR IF 9 OR LESS
FB7F C1 11           CMP B      #$11
FB81 2B 56           BMI      CONERR  NOT HEX IF < A
FB83 C1 16           CMP B      #$16
FB85 2E 52           BGT      CONERR  NOT HEX IF > F
FB87 C0 07           SUB B      #7      MOVE A-F ABOVE 0-9

```

*SHIFT LO & HI BYTES LEFT 4 BITS
 FB89 8D 54 HEXN1 BSR SHIFT2
 FB8B 8D 52 BSR SHIFT2

FB8D FA 7014 ORA B NBRLO
 FB90 F7 7014 STA B NBRLO

FB93 20 CB BRA NUMLUP

* INPUT A DECIMAL NUMBER
 *TEST FOR A LEGAL DIGIT
 FB95 C1 09 DECNUM CMP B #\$09
 FB97 2E 40 BGT CONERR NOT DECIMAL IF > 9

*MULTIPLY SAVED VALUE BY 10 & ADD IN NEW DIGIT
 *NOTE THAT 10X=2X+8X
 *MULTIPLY CURRENT NUMBER BY 2 TO GET 2X VALUE
 FB99 8D 49 BSR SHIFT
 *SAVE THIS *2 NUMBER TEMPORARILY
 FB9B FE 7013 LDX NBRHI
 FB9E FF 70DC STX NBR2X
 *MULTIPLY THIS # BY 4 TO GET 8X VALUE
 FBA1 8D 3C BSR SHIFT2
 *(NBRHI,NBRLO) NOW HOLDS *8
 *GENERATE DIGIT+8X+2X

FBA3 4F	CLR A	ACCA WILL HOLD MS BYTE
FBA4 FB 70DD	ADD B NBR2X+1	ADD 2X LS BYTE TO DIGIT
FBA7 B9 70DC	ADC A NBR2X	ADD 2X MS BYTE
FBAA 25 2D	BCS CONERR	CARRY OUT OF MS BYTE IS AN ERROR
FBAC FB 7014	ADD B NBRLO	ADD 8X LS BYTE
FBAF B9 7013	ADC A NBRHI	ADD 8X MS BYTE
FBB2 25 25	BCS CONERR	CARRY OUT OF MS BYTE IS AN ERROR
FBB4 F7 7014	STA B NBRLO	SAVE FINAL LS BYTE
FBB7 B7 7013	STA A NBRHI	SAVE FINAL MS BYTE

FBBA 20 A4 BRA NUMLUP

* INPUT AN OCTAL NUMBER
 FBBC C1 07 OCTNUM CMP B #\$07
 FBBE 2E 19 BGT CONERR NOT OCTAL IF > 7

*SHIFT HI & LO BYTES 3 PLACES LEFT - CARRY OUT OF HI
 *BYTE IS ILLEGAL

FBC0 8D 1D	BSR SHIFT2	
FBC2 8D 20	BSR SHIFT	

FBC4 FA 7014 ORA B NBRLO ADD IN NEW DIGIT
 FBC7 F7 7014 STA B NBRLO

FBCA 7E FB60 JMP NUMLUP

```

*****
*GOOD NUMBER - SCAN WAS SUCCESSFUL
*UPDATE GOOD SYNTAX LINE POINTER
FBCD FE 700C GUDNUM LDX LINPTR
FBD0 FF 700A . STX SYNPTR
FBD3 FE 70D2 LDX XTEMP2
FBD6 86 01 LDA A #1 SET "GOOD SCAN" FLAG
FBD8 39 RTS

*****
*CONVERSION ERROR - SCAN WAS UNSUCCESSFUL
FBD9 FE 70D2 CONERR LDX XTEMP2
FBDC 4F CLR A
FBDD 4A DEC A
FBDE 39 RTS
*-----
*SHIFT LEFT 2 POSITIONS
FBDF 8D 03 SHIFT2 BSR SHIFT
FBE1 8D 01 BSR SHIFT
FBE3 39 RTS
*-----
*SHIFT A TWO BYTE NUMBER LEFT ONE POSITION
FBE4 78 7014 SHIFT ASL NBRLO
FBE7 79 7013 ROL NBRHI
FBEA 25 01 BCS SHFTER
FBEC 39 RTS
*ERROR - HI ORDER BYTE OVERFLOW
*ABORT NUMBER ROUTINE DIRECTLY THRU STACK ADJ. & A JUMP
FBED 31 SHFTER INS
FBEE 31 INS
FBEF 20 E8 BRA CONERR

*****
*=====
*OUTPUT A SPACE
FBFI 86 20 OUTSP LDA A #$20
FBF3 BD FE76 JSR OUTCHR
FBF6 39 RTS

*****
*=====
*OUTPUT AN "=" SIGN
FBF7 86 3D OUTEQ LDA A #'=
FBF9 BD FE76 JSR OUTCHR
FBFC 39 RTS

*****
*=====
*OUTPUT A 1 BYTE NUMBER
*ACCA, ACCB, & IX ARE PRESERVED
FBFD 37 OUTIBY PSH B
FBFE C6 01 LDA B #1
FC00 8D 09 BSR OUTNUM
FC02 33 PUL B
FC03 39 RTS

```

```

=====
*OUTPUT A 2 BYTE NUMBER
*ACCA, ACCB, & IX ARE PRESERVED
FC04 37 OUT2BY PSH B
FC05 C6 02 LDA B #2
FC07 8D 02 BSR OUTNUM
FC09 33 PUL B
FC0A 39 RTS

=====
*DISPLAY THE NUMBER POINTED AT BY THE ADDRESS IN THE
* INDEX REGISTER
* AND OUTPUT IT ACCORDING TO THE BASE SPECIFIED IN
* "DBCODE"
*LEADING ZEROES ARE INCLUDED
*ACCA & IX ARE PRESERVED
*ACCB IS INPUT AS THE NUMBER OF BYTES COMPRISING THE
* NUMBER
*GLOBAL VARIABLES FOR EXTERNAL COMMUNICATION
*IBCODE - INPUT BASE CODE
*DBCODE - DISPLAY BASE CODE

*LOCAL VARIABLES
*DECDIG - DECIMAL DIGIT BEING BUILT
*NUMBHI - HI BYTE OF NUMBER BEING OUTPUT
*NUMBLO - LO BYTE OF NUMBER BEING OUTPUT

```

```

FC0B FF 70D0 OUTNUM STX XTEMP1
FC0E 36 PSH A
FC0F EE 00 LDX X GET THE TWO BYTES AT THAT
* ADDRESS
FC11 FF 70DA STX NUMBHI PUT THEM IN A SCRATCH AREA FOR
* PROCESSING
FC14 B6 7011 LDA A DBCODE GET DISPLAY BASE

```

```

FC17 81 01 CMP A #1
FC19 27 0C BEQ OUTHEX
FC1B 81 02 CMP A #2
FC1D 27 1E BEQ OUTDEC
FC1F 81 03 CMP A #3
FC21 27 5E BEQ OUTOCT
FC23 81 04 CMP A #4
FC25 27 78 BEQ OUTBIN

```

```

*****
*OUTPUT A HEX NUMBER
FC27 58 OUTHEX ASL B 1 BYTE=2 CHARS, 2 BYTES=4 CHARS
*GET NEXT 4 BITS

```

```

FC28 BD FCB3 DISNU1 JSR LSH2
FC2B BD FCB3 JSR LSH2

```

```

FC2E 84 0F AND A #$F EXTRACT 4 BITS

```

FC30 81 09	CMP A #9	
FC32 2F 02	BLE DISNU2	
FC34 8B 07	ADD A #7	CONVERT 10:15 TO A:F

FC36 8D 75	DISNU2 BSR OUTIT	
FC38 5A	DEC B	
FC39 26 ED	BNE DISNU1	
FC3B 20 35	BRA OUTDE5	

*OUTPUT A DECIMAL NUMBER

FC3D 5A	OUTDEC DEC B	TEST # OF BYTES TO OUTPUT
FC3E 27 0B	BEQ OUTDE1	

*INITIALIZE FOR OUTPUT OF A 2 BYTE NUMBER

FC40 CE FC77	LDX #C10K	
FC43 B6 70DA	LDA A NUMBH1	
FC46 F6 70DB	LDA B NUMBLO	
FC49 20 07	BRA OUTDE2	

*INITIALIZE FOR OUTPUT OF A 1 BYTE NUMBER

FC4B CE FC7B OUTDE1 LDX #C100		
FC4E 4F CLR A		
FC4F F6 70DA LDA B NUMBH1		

FC52 7F 70D9 OUTDE2 CLR	DEC DIG	CLEAR THE DIGIT TO OUTPUT
-------------------------	---------	---------------------------

*SUBTRACT THE POWER OF 10 CONVERSION CONSTANT

FC55 E0 01 OUTDE3 SUB B 1,X		
FC57 A2 00 SBC A 0,X		
FC59 25 05 BCS OUTDE4	TEST FOR BORROW (CARRY)	

FC5B 7C 70D9 INC	DEC DIG	NO BORROW YET - INC DIGIT BEING
*		BUILT
FC5E 20 F5 BRA	OUTDE3	REPEAT LOOP
*		BORROW GENERATED - CANCEL LAST SUBTRACTION
FC60 EB 01 OUTDE4 ADD B 1,X		
FC62 A9 00 ADC A 0,X		

*BUILDING OF DIGIT TO OUTPUT IS COMPLETE - PRINT IT

FC64 36 PSH A		SAVE LO BYTE OF NUMBER BEING
*		OUTPUT

FC65 B6 70D9 LDA A	DEC DIG	GET DIGIT
--------------------	---------	-----------

FC68 8D 43 BSR	OUTIT	PRINT IT
----------------	-------	----------

FC6A 32 PUL A		RESTORE LO BYTE
---------------	--	-----------------

*GET NEXT LOWER POWER OF 10

FC6B 08 INX		
-------------	--	--

FC6C 08 INX		
-------------	--	--

FC6D 8C FC81 CPX #C10K+10	ARE WE THRU WITH UNITS	
*	CONVERSION?	

FC70 26 E0 BNE OUTDE2	IF NOT, BACK TO GET NEXT DIGIT	
-----------------------	--------------------------------	--

FC72 32 OUTDE5 PUL A	IF YES, RESTORE REGISTERS & RETU	
----------------------	----------------------------------	--

FC73 FE 70D0	LDX XTEMP1
FC76 39	RTS
*DECIMAL OUTPUT CONVERSION CONSTANTS	
FC77 2710	C10K FDB 10000
FC79 03E8	FDB 1000
FC7B 0064	C100 FDB 100
FC7D 000A	FDB 10
FC7F 0001	FDB 1

*OUTPUT AN OCTAL NUMBER	
*FIRST DIGIT IS A ONE OR A ZERO	
FC81 58	OUTOCT ASL B FIRST APPROXIMATION OF # OF * DIGITS TO OUTPUT
FC82 4F	CLR A
FC83 C1 02	CMP B #2
FC85 2E 06	BGT OUTOC1
FC87 8D 2A	BSR LSH2 1 BYTE - GET FIRST 2 BITS
FC89 8D 22	BSR OUTIT
FC8B 20 05	BRA DISNU3 GO OUTPUT LAST 2 DIGITS
FC8D 8D 29	*TWO BYTE # - OUTPUT HI ORDER BIT/DIGIT
FC8F 8D 1C	OUTOC1 BSR LEFSHF
FC91 5C	BSR OUTIT INC B 5 MORE DIGITS TO GO
FC92 8D 1F	DISNU3 BSR LSH2
FC94 8D 22	BSR LEFSHF
FC96 84 07	AND A #7 EXTRACT 3 BITS
FC98 8D 13	BSR OUTIT
FC9A 5A	DEC B COUNT THIS DIGIT
FC9B 26 F5	BNE DISNU3 ARE WE DONE?
FC9D 20 D3	BRA OUTDE5 YES

*OUTPUT A BINARY NUMBER	
FC9F 58	OUTBIN ASL B
FCA0 58	ASL B
FCA1 58	ASL B
*GET NEXT BIT	
FCA2 8D 14	DISNU4 BSR LEFSHF
FCA4 84 01	AND A #1 EXTRACT THE BIT
FCA6 8D 05	BSR OUTIT OUTPUT IT
FCA8 5A	DEC B COUNT IT
FCA9 26 F7	BNE DISNU4 ARE WE DONE?
FCA9 20 C5	BRA OUTDE5 YES

*CONVERT TO A NUMERIC ASCII DIGIT & OUTPUT IT	
FCAD 8B 30	OUTIT ADD A #\$30
FCAF BD FE76	JSR OUTCHR
FCB2 39	RTS

*LEFT SHIFT 2 BITS	

FCB3 8D 03	LSH2	BSR	LEFSHF
FCB5 8D 01		BSR	LEFSHF
FCB7 39		RTS	

* LEFT SHIFT THE 3 BYTE NUMBER 1 BIT

FCB8 78 70DB	LEFSHF	ASL	NUMBLO
FCBB 79 70DA		ROL	NUMBH1
FCBE 49		ROL A	
FCBF 39		RTS	

*=====

* THIS ROUTINE GETS THE NEXT CHARACTER FROM THE INPUT
* LINE BUFFER

*ACCA IS PRESERVED

*ACCB IS LOADED WITH THE CHARACTER

* IX IS INCREMENTED & LEFT POINTING TO THE CHARACTER
* RETURNED

FCC0 FE 700C	GETCHR	LDX	LINPTR
FCC3 08		INX	
FCC4 E6 00		LDA B	X
FCC6 FF 700C		STX	LINPTR
FCC9 7F 700E		CLR	BOLFLG
	*		SET FLAG TO NOT AT "BEGINNING OF LINE"
FCCC 39		RTS	

*=====

* THIS ROUTINE GETS THE NEXT CHARACTER IN THE COMMAND
* LISTS

*ACCA IS THE CHARACTER RETRIEVED

*ACCB IS PRESERVED

* IX IS INCREMENTED & LEFT POINTING TO THE CHARACTER
* RETURNED

FCCD FE 70D7	GETLST	LDX	LISPTR	GET CURRENT LIST POINTER
FCD0 08		INX		MOVE POINTER TO NEXT CHAR
FCD1 A6 00		LDA A	X	GET CHARACTER POINTED AT
FCD3 FF 70D7		STX	LISPTR	SAVE POINTER
FCD6 39		RTS		AND RETURN

*=====

* COMMAND LISTS

* A CARRIAGE RETURN SIGNIFIES END-OF-COMMAND

* A LINE FEED SIGNIFIES END-OF-COMMAND-LIST

* LIST 1 - MAJOR COMMANDS

FCDF 75 2	COMLST	EQU	*	
		FCC	'REG'	DISPLAY REGISTERS
FCD8 45				
FCD9 47				
FCDA 0D		FCB	CR	
FCDB 47		FCC	'GOTO'	GO TO MEMORY ADDRESS
FCDC 4F				
FCDD 54				
FCDE 4F				
FCDF 0D		FCB	CR	
FCE0 53		FCC	'SEI'	SET INTERRUPT MASK

FCE1	45		
FCE2	49		
FCE3	0D	FCB	CR
FCE4	43	FCC	'CLI' CLEAR INTERRUPT MASK
FCE5	4C		
FCE6	49		
FCE7	0D	FCB	CR
FCE8	43	FCC	'COPY' COPY FROM ONE LOCATION TO
FCE9	4F		
FCEA	50		
FCEB	59		
	*		ANOTHER
FCEC	0D	FCB	CR
FCED	42	FCC	'BREAK' SET BREAKPOINT (SWI CODE)
FCEE	52		
FCEF	45		
FCF0	41		
FCF1	4B		
FCF2	0D	FCB	CR
FCF3	49	FCC	'IBASE' SET INPUT BASE
FCF4	42		
FCF5	41		
FCF6	53		
FCF7	45		
FCF8	0D	FCB	CR
FCF9	44	FCC	'DBASE' SET DISPLAY BASE
FCFA	42		
FCFB	41		
FCFC	53		
FCFD	45		
FCFE	0D	FCB	CR
FCFF	43	FCC	'CONTINUE' CONTINUE FROM "SWI"
FD00	4F		
FD01	4E		
FD02	54		
FD03	49		
FD04	4E		
FD05	55		
FD06	45		
FD07	0D	FCB	CR
FD08	44	FCC	'DISPLAY' DISPLAY MEMORY DATA
FD09	49		
FD0A	53		
FD0B	50		
FD0C	4C		
FD0D	41		
FD0E	59		
FD0F	0D	FCB	CR
FD10	53	FCC	'SET' SET MEMORY DATA
FD11	45		
FD12	54		
FD13	0D	FCB	CR
FD14	56	FCC	'VERIFY' VERIFY THAT MEMORY
FD15	45		
FD16	52		
FD17	49		

FD18 46			
FD19 59	*		CONTENT IS UNCHANGED
FD1A 0D	FCB	CR	
FD1B 53	FCC	'SEARCH'	SEARCH MEMORY FOR A
FD1C 45			
FD1D 41			
FD1E 52			
FD1F 43			
FD20 48	*		BYTE STRING
FD21 0D	FCB	CR	
FD22 54	FCC	'TEST'	TEST A RANGE OF MEMORY
FD23 45			
FD24 53			
FD25 54			
FD26 0D	FCB	CR	
FD27 49	FCC	'INT'	SET INTERRUPT POINTER
FD28 4E			
FD29 54			
FD2A 0D	FCB	CR	
FD2B 4E	FCC	'NMI'	SET NON-MASKABLE
FD2C 4D			
FD2D 49	*		INTERRUPT POINTER
FD2E 0D	FCB	CR	
FD2F 53	FCC	'SWI'	SET SOFTWARE INTERRUPT
FD30 57			
FD31 49	*		POINTER
FD32 0D	FCB	CR	
FD33 43	FCC	'COMPARE'	PRINT SUM & DIFFERENCE
FD34 4F			
FD35 4D			
FD36 50			
FD37 41			
FD38 52			
FD39 45	*		OF 2 NUMBERS
FD3A 0D	FCB	CR	
FD3B 44	FCC	'DUMP'	DUMP MEMORY IN MIKBUG OR IMAGE
FD3C 55			
FD3D 4D			
FD3E 50	*		FORMAT
FD3F 0D	FCB	CR	
FD40 4C	FCC	'LOAD'	LOAD MIKBUG TAPE
FD41 4F			
FD42 41			
FD43 44			
FD44 0D	FCB	CR	
FD45 44	FCC	'DELAY'	DELAY SPECIFIED # OF BYTES
FD46 45			
FD47 4C			
FD48 41			
FD49 59			

FD4A 0D FCB CR
FD4B 0A FCB LF END OF LIST 1
*LIST 2 - MODIFIER TO DUMP
FD4C 54 FCC 'TO' DESTINATION ACIA
FD4D 4F
FD4E 0D FCB CR
FD4F 0A FCB LF END OF LIST 2

*LIST 3 - NUMBER BASE SPECIFIERS

FD50 48 FCC 'HEX' BASE 16
FD51 45
FD52 58
FD53 0D FCB CR
FD54 44 FCC 'DEC' BASE 10
FD55 45
FD56 43
FD57 0D FCB CR
FD58 4F FCC 'OCT' BASE 8
FD59 43
FD5A 54
FD5B 0D FCB CR
FD5C 42 FCC 'BIN' BASE 2
FD5D 49
FD5E 4E
FD5F 0D FCB CR
FD60 0A FCB LF END OF LIST 3
*LIST 4 - INFORMATION REQUEST
FD61 3F FCC '?'
FD62 0D FCB CR
FD63 0A FCB LF END OF LIST 4

*LIST 5 - REGISTER NAMES

FD64 2E FCC '.CC'
FD65 43
FD66 43
FD67 0D FCB CR
FD68 2E FCC '.B'
FD69 42
FD6A 0D FCB CR
FD6B 2E FCC '.A'
FD6C 41
FD6D 0D FCB CR
FD6E 2E FCC '.IX'
FD6F 49
FD70 58
FD71 0D FCB CR
FD72 2E FCC '.PC'
FD73 50
FD74 43
FD75 0D FCB CR
FD76 2E FCC '.SP'
FD77 53

FD78 50
FD79 0D FCB CR
FD7A 0A FCB LF END OF LIST 5

*LIST 6 - MODIFIERS TO "DISPLAY"
FD7B 44 FCC 'DATA'
FD7C 41
FD7D 54
FD7E 41
FD7F 0D FCB CR
FD80 55 FCC 'USED'
FD81 53
FD82 45
FD83 44
FD84 0D FCB CR
FD85 0A FCB LF END OF LIST 6

*LIST 7 - MODIFIER TO "LOAD"
FD86 46 FCC 'FROM' SOURCE ACIA
FD87 52
FD88 4F
FD89 4D
FD8A 0D FCB CR
FD8B 0A FCB LF END OF LIST 6

* THIS ROUTINE CONSTRUCTS A LINE OF INPUT BY GETTING
* ALL INPUT
* CHARACTERS UP TO AND INCLUDING A CARRIAGE RETURN
* (WHICH THEN
* DESIGNATES "END OF LINE").
* TYPING RUBOUT WILL DELETE THE PREVIOUS CHARACTER
* TYPING CONTROL-C WILL ABORT THE LINE
* TYPING CONTROL-Z WILL USE THE PREVIOUS LINE
* THE INPUT LINE IS STORED BEGINNING AT THE ADDRESS
* STORED IN BUFBE
* AND ENDING AT THE ADDRESS STORED IN BUFE
*ACCA, ACCB, & IX ARE NOT PRESERVED
*
*GLOBAL VARIABLES
*BUFBE - INPUT LINE START OF BUFFER
*BUFE - INPUT LINE END OF BUFFER

*LOCAL CONSTANTS
005C BAKSLA EQU 92 A BACKSLASH
007F DELETE EQU 127 CODE TO DELETE THE PREVIOUS
* CHARACTER
*
*

***** ROUTINE ENTRY POINT

FD8C FE 702C	GETLIN	LDX	BUFBEG	SET POINTER TO ONE LESS THAN THE BEGINNING OF THE LINE BUFFE R
	*			
	*			
FD8F 5F		CLR B		ACCB HOLDS LAST INPUT CHAR
	*			
FD90 BC 702E	NXTCHR	CPX	BUFEND	CHECK CURRENT LINE END AGAINST BUFFER END
	*			
FD93 26 09		BNE	GETIT	
				* LINE TOO LONG - ABORT IT AS IF A CONTROL-C HAD BEEN TYPED
FD95 CE FF07		LDX	#MSGTL	GET MESSAGE
FD98 BD FE4B		JSR	OUTSTR	OUTPUT IT
FD9B C6 03		LDA B	#3	PUT CTL-C IN ACCB
FD9D 39		RTS		
FD9E BD FE59	GETIT	JSR	INPCHR	GET A CHARACTER (RETURNED IN ACCA)
	*			
FDA1 84 7F		AND A	#127	DROP PARITY BIT
				*CONTROL-Z COPIES FROM PRESENT POSITION TO PREVIOUS END OF LINE
	*			
FDA3 81 1A		CMP A	#26	IS CHAR A CONTROL-Z?
FDA5 26 04		BNE	TSTCR	
FDA7 BD FEC7		JSR	DOCRLF	YES, TYPE CR-LF
FDA9 39		RTS		
FDAB 81 0D	TSTCR	CMP A	#13	IS CHAR A CR?
FDAD 27 04		BEQ	TSTCR1	
FDAF 81 0A		CMP A	#10	OR A LF?
FDB1 26 0C		BNE	NOTEOL	
FDB3 08	TSTCR1	INX		
FDB4 A7 00		STA A	X	YES, STORE THE TERMINATOR
FDB6 7D 7029		TST	HDXFLG	TEST FOR HALF-DUPLEX TERMINAL
FDB9 26 03		BNE	TSTCR2	
FDBB BD FEC7		JSR	DOCRLF	TYPE CR-LF
FDBE 39	TSTCR2	RTS		NOW RETURN
	*			
FDBF 81 03	NOTEOL	CMP A	#3	IS CHAR A CONTROL-C?
FDC1 26 07		BNE	NOTCTC	
	*	ECHO AN UP-ARROW		
FDC3 16		TAB		RETURN CONTROL-C IN ACCB
FDC4 86 5E		LDA A	#'1	
FDC6 BD FE76		JSR	OUTCHR	
FDC9 39		RTS		
FDCA 81 7F	NOTCTC	CMP A	#DELETE	NO, IS IT DELETE?
FDCC 27 25		BEQ	RUBNOW	IF YES, GO TO RUBNOW
				*CONVERT LOWER CASE TO UPPER CASE
FDCE 81 60		CMP A	#\$60	BELLOW L.C. A?

FDD0 23 06	BLS	STORIT	
FDD2 81 7A	CMP A	#\$7A	ABOVE L.C. Z?
FDD4 22 02	BHI	STORIT	
FDD6 80 20	SUB A	#32	CONV L.C. ALPHABETIC TO U.C.
FDD8 08	STORIT	INX	NOT A DELETE, SO ADVANCE TO
	*		NEXT CHARACTER
FDD9 A7 00	STA A	X	STORE IT IN INPLIN
FDBB C1 7F	CMP B	#DELETE	IS LAST CHAR A DELETE?
FDDD 27 03	BEQ	OUTBAK	IF SO, GO TO OUTBAK
FDDF 16	TAB		ITS NOT, UPDATE LAST CHAR
FDE0 20 07	BRA	ECHO	GO ECHO IT
	*		
	*	LAST CHAR WAS A DELETE, BUT THIS ONE ISN'T	
FDE2 16	OUTBAK	TAB	UPDATE LAST CHAR
FDE3 86 5C	LDA A	#BAKSLA	PRINT A -
FDE5 BD FE76	JSR	OUTCHR	BACKSLASH
FDE8 17	TBA		RESTORE CURRENT CHAR FOR ECHO
FDE9 7D 7029	ECHO	TST	HDXFLG TEST FOR HALF DUPLEX TERMINAL
FDEC 26 03	BNE	ECHO1	
FDEE BD FE76	JSR	OUTCHR	NOW ECHO IT
FDF1 20 9D	ECHO1	BRA	NXTCHR GET ANOTHER
	*		
	*	CURRENT CHARACTER IS A DELETE	
	*	TEST LINE LENGTH - IF ITS ZERO, IGNORE THIS DELETE	
	*	SINCE	
	*	WE CAN'T DELETE PRIOR TO FIRST CHARACTER IN INPUT	
	*	LINE	
FDF3 BC 702C	RUBNOW	CPX	BUFBEG
FDF6 27 98		BEQ	NXTCHR
FDF8 C1 7F		CMP B	#DELETE WAS LAST CHAR A DELETE?
Fdfa 27 06		BEQ	LASWAS
	*	LAST CHAR WASN'T A DELETE	
FDFC 16	TAB		UPDATE LAST CHAR (WITH A DELETE)
FDFD 86 5C	LDA A	#BAKSLA	PRINT A -
FDFE BD FE76	JSR	OUTCHR	BACKSLASH
	*	LAST CHAR WAS A DELETE	
FE02 A6 00	LASWAS	LDA A X	GET THE CHAR TO BE DELETED
FE04 09	\	DEX	DECREMENT LINE POINTER
FE05 20 E2		BRA	ECHO ECHO DELETED CHARACTER

*=====

* INITIALIZATION ROUTINE

* DISABLE INTERRUPTS

FE07 0F	SEI		
FE08 86 01	INITAL	LDA A #1	
FE0A B7 7010	STA A	IBCODE	SET INPUT BASE TO HEX
FE0D B7 7011	STA A	DBCODE	SET DISPLAY BASE TO HEX
	*	SET UP DISPLAY BASE NUMBER	
FE10 86 10	LDA A	#16	

```

FE12 B7 7012 STA A DBNBR
*MAX # OF CHARACTERS PER LINE
FE15 86 48 LDA A #72
FE17 B7 702B STA A CPLMAX
FE1A 7F 7023 CLR INPFLG DEFAULT INPUT FROM THE TERMINAL
FE1D 7F 7026 CLR OUTFLG DEFAULT OUTPUT TO THE TERMINAL
FE20 7F 7029 CLR HDXFLG CLEAR HALF-DUPLEX FLAG
*INITIALIZE ACIA1 & ACIA2 TO 7 BITS & EVEN PARITY
*RESET BOTH
FE23 86 03 LDA A #3
FE25 B7 7F42 STA A ACIA1-1
FE28 B7 7F44 STA A ACIA2-1
*SET EM UP
FE2B 86 02 LDA A #2
FE2D B7 7F42 STA A ACIA1-1
FE30 B7 7F44 STA A ACIA2-1
*SET UP SWI INTERRUPT ADDRESS POINTER
FE33 CE F501 LDX #TYPWSWI TYPE "SWI" & DO "REG" COMMAND
FE36 FF 7004 STX SWIVEC
*INITIALIZE TO MONDEB'S COMMAND LISTS
FE39 CE FCD6 LDX #COMLST-1
FE3C FF 7008 STX COMADR
*TIME CONSTANT FOR A 2 MICROSECOND CLOCK
FE3F 86 53 LDA A #83
FE41 B7 70DE STA A TIMCON
*ALLOW TIME FOR TTY MOTOR TO COME UP TO SPEED
FE44 CE 01F4 LDX #500
FE47 BD F9BD JSR TIMDEL
FE4A 39 RTS
*=====
*OUTPUT A CHARACTER STRING WHICH BEGINS AT THE ADDRESS
*           IN THE INDEX REGISTER
*ACCA & ACCB ARE PRESERVED
*IX IS LEFT POINTING TO THE STRING TERMINATOR
FE4B 36 OUTSTR PSH A
FE4C A6 00 OUTSTI LDA A X GET CHAR POINTED TO
FE4E 81 04 CMP A #4 IS IT A STRING TERMINATOR?
FE50 27 05 BEQ OUTEND DONE IF IT IS
FE52 8D 22 BSR OUTCHR ISN'T, OUTPUT IT
FE54 08 INX ON TO NEXT CHARACTER
FE55 20 F5 BRA OUTSTI
FE57 32 OUTEND PUL A
FE58 39 RTS RETURN

*=====
*INPUT A CHARACTER FROM AN ACIA TO ACCA
*IF INPFLG = 0, INPUT IS FROM TERMINAL ACIA
*IF INPFLG = 1, INPUT IS FROM ANY ACIA
*ACCB & IX ARE PRESERVED
FE59 FF 70CE INPCHR STX XTEMP SAVE IX
FE5C 7D 7023 TST INPFLG TEST INPUT SOURCE FLAG
FE5F 26 05 BNE INPCH1
*INPFLG=0: INPUT FROM TERMINAL ACIA
FE61 CE 7F43 LDX #ACIA1

```

```

FE64 20 03          BRA     INPCH2
                    * INPFLG=1: INPUT FROM ANY ACIA
FE66 FE 7024 INPCH1 LDX     INPADR   GET ITS ADDRESS
FE69 09           INPCH2 DEX     POINT TO CONTROL REG
FE6A A6 00           INPCH3 LDA A  X      GET ACIA STATUS BYTE
FE6C 85 01           BIT A  #1      CHAR WAITING?
FE6E 27 FA          BEQ     INPCH3   IF NOT, TRY AGAIN
FE70 A6 01          LDA A  1,X    YES, GET IT
FE72 FE 70CE          LDX     XTEMP    RESTORE IX
FE75 39           RTS

*=====
*OUTPUT THE CHARACTER IN ACCA TO THE DESIRED OUTPUT
* DEVICE/LOCATION
* IF OUTFLG = 0, OUTPUT IS TO TERMINAL
* IF OUTFLG = 1, OUTPUT IS TO ACIA ADDRESS STORED IN
* OUTADR
* IF OUTFLG = 2, OUTPUT IS TO ADDRESS IN OUTADR & THIS
* ADDR IS THEN INCREMENTED
*ACCA, ACCB, & IX ARE PRESERVED
FE76 37          OUTCHR PSH B  SAVE ACCB
FE77 7D 7026          TST     OUTFLG  TEST OUTPUT DESTINATION FLAG
FE7A 27 21          BEQ     OUTCH4  SKIP THIS CODE IF TERMINAL
*                         OUTPUT

*OUTPUT TO SOMETHING OTHER THAN TERMINAL
FE7C FF 70CE          STX     XTEMP    SAVE IX
FE7F FE 7027          LDX     OUTADR   GET OUTPUT CHAR DESTINATION
*                         ADDRESS
FE82 C6 02           LDA B  #2
FE84 F1 7026           CMP B  OUTFLG
FE87 27 09           BEQ     OUTCH2

*OUTFLG = 1: ANY ACIA OUTPUT
FE89 09           DEX     POINT TO ACIA STATUS REG
FE8A E5 00           OUTCH1 BIT B  X      TEST TDRE BIT
FE8C 27 FC           BEQ     OUTCH1  LOOP IF NOT READY TO ACCEPT A
*                         NEW CHAR
FE8E A7 01           STA A  1,X    NOW READY - SEND IT
FE90 20 06           BRA     OUTCH3

*OUTFLG = 2: MEMORY OUTPUT
FE92 A7 00           OUTCH2 STA A  X      SAVE CHAR IN MEMORY
FE94 08           INX
FE95 FF 7027           STX     OUTADR   UPDATE OUTPUT ADDRESS

FE98 FE 70CE OUTCH3 LDX     XTEMP    RESTORE IX
FE9B 33           PUL B   RESTORE ACCB
FE9C 39           RTS

*OUTFLG = 0: TERMINAL ACIA OUTPUT
*IGNORE LINE FEEDS
FE9D 81 0A           OUTCH4 CMP A  #LF
FE9F 26 02           BNE     OUTCH5

```

```

FEA1 33          PUL B
FEA2 39          RTS

FEA3 81 0D      OUTCH5 CMP A #CR      TEST FOR CARRIAGE RETURN
FEA5 26 04      BNE    OUTCH6
FEA7 8D 1E      BSR    DOCRLF
FEA9 33          PUL B
FEAA 39          RTS

FEAB F6 702A OUTCH6 LDA B CPLCNT   GET "CHARACTERS PER LINE" COUNT
FEAE F1 702B     CMP B CPLMAX    COMPARE TO MAX PERMISSIBLE
FEB1 2C 0B      BGE    OUTCH7    SEND CR-LF IF GREATER
*LESS THAN MAX, BUT ALSO SEND CR-LF IF 10 FROM END AND
*PRINTING A SPACE
FEB3 CB 0A      ADD B #10
FEB5 F1 702B     CMP B CPLMAX
FEB8 2D 06      BLT    OUTCH8
FEBA 81 20      CMP A #$20    NEAR END, TEST IF ABOUT TO
*PRINT A SPACE
FEBc 26 02      BNE    OUTCH8

*TERMINAL LINE FULL OR NEARLY FULL - INTERJECT A CR-LF
FEBE 8D 07      OUTCH7 BSR    DOCRLF
FEC0 7C 702A OUTCH8 INC    CPLCNT   BUMP COUNTER
FEC3 8D 20      BSR    TOACIA   SEND IT TO ACIA1
FEC5 33          PUL B
FEC6 39          RTS

=====
*SEND A CARRIAGE RETURN-LINE FEED TO THE TERMINAL
*ACCA, ACCB, & IX ARE PRESERVED
FEC7 36          DOCRLF PSH A
FEC8 37          PSH B
FEC9 86 0D      LDA A #CR
FECB 8D 18      BSR    TOACIA
FECd 86 0A      LDA A #LF
FECf 8D 14      BSR    TOACIA

*ALLOW TIME FOR THE CARRIAGE TO RETURN BY SENDING NULL
*CHARACTERS
*SEND 1 NULL PER 16 CHARACTERS
*DIVIDE CPLCNT BY 16
FEDI F6 702A     LDA B CPLCNT
FED4 54          LSR B
FED5 54          LSR B
FED6 54          LSR B
FED7 54          LSR B
FED8 5C          INC B      ALWAYS SEND AT LEAST 1 NULL
FED9 4F          DOCRL1 CLR A      GET A NULL
FEDA 8D 09      BSR    TOACIA   SEND IT
FEDC 5A          DEC B
FEDD 26 FA      BNE    DOCRL1
FEDF 7F 702A     CLR    CPLCNT   ZERO "CHARACTERS PER LINE" COUNT

```

```

FEE2 33          PUL B
FEE3 32          PUL A
FEE4 39          RTS

=====
*PUT CHAR IN ACCA INTO TERMINAL ACIA
*ACCA, ACCB, & IX ARE PRESERVED
FEE5 36          TOACIA PSH A           SAVE CHAR
FEE6 86 02        LDA A   #2           GET ACIA TRANSMIT REG STATUS BIT
FEE8 B5 7F42      TOACII BIT A ACIA1-1  REGISTER EMPTY?
FEEB 27 FB        BEQ    TOACII       IF NOT, LOOP BACK
FEED 32          PUL A             YES, RESTORE CHARACTER
FEEE B7 7F43      STA A  ACIA1       SEND IT
FEFI 39          RTS

=====
*MISC TEXT
FEF2 4D          MSGHED FCC      'MONDEB 1.00' MONITOR HEADER TYPEOUT
FEF3 4F
FEF4 4E
FEF5 44
FEF6 45
FEF7 42
FEF8 20
FEF9 31
FEFA 2E
FEFB 30
FEFC 30
FEFD 0D          FCB     CR, 4
FEFE 04
FEFF 2A          MSGPRM FCB     '* , 4      PROMPT STRING
FF00 04
FF01 0D          MSGSWI FCB     CR
FF02 53          FCC     ' SWI : '
FF03 57
FF04 49
FF05 3A
FF06 04          FCB     4
FF07 54          MSGLTL FCC      ' TOO LONG' TYPED IF INPUT LINE IS
FF08 4F
FF09 4F
FF0A 20
FF0B 4C
FF0C 4F
FF0D 4E
FF0E 47

*                      TOO LONG
FF0F 04          FCB     4
FF10 4E          MSGNBR FCC      ' NOT SET' BREAK NOT SET
FF11 4F
FF12 54
FF13 20
FF14 53
FF15 45
FF16 54
FF17 04          FCB     4
FF18 53          MSGBAT FCC      ' SET @ ' BREAK AT -

```

FF19	45		
FF1A	54		
FF1B	20		
FF1C	40		
FF1D	20		
FF1E	04	FCB	4
FF1F	4F	MSGVER FCC	'OK' CHECKSUM VERIFIES
FF20	4B		
FF21	04	FCB	4
FF22	43	MSGNVE FCC	'CHECKSUM ERROR ' FOR VERIFY &
FF23	48		
FF24	45		
FF25	43		
FF26	4B		
FF27	53		
FF28	55		
FF29	4D		
FF2A	20		
FF2B	45		
FF2C	52		
FF2D	52		
FF2E	4F		
FF2F	52		
FF30	20		
*			
LOAD COMMANDS			
FF31	04	FCB	4
FF32	43	MSGCCL FCC	'CANT CLEAR' TEST COMMAND
FF33	41		
FF34	4E		
FF35	54		
FF36	20		
FF37	43		
FF38	4C		
FF39	45		
FF3A	41		
FF3B	52		
FF3C	04	FCB	4
FF3D	43	MSGCSO FCC	'CANT SET TO ONES' TEST COMMAND
FF3E	41		
FF3F	4E		
FF40	54		
FF41	20		
FF42	53		
FF43	45		
FF44	54		
FF45	20		
FF46	54		
FF47	4F		
FF48	20		
FF49	4F		
FF4A	4E		
FF4B	45		
FF4C	53		
FF4D	04	FCB	4
FF4E	53	MSGISIS FCC	'SUM IS ' COMPARE COMMAND
FF4F	55		

FF50 4D
FF51 20
FF52 49
FF53 53
FF54 20
FF55 04 FCB 4
FF56 2C MSGDIS FCC ', DIF IS ' COMPARE COMMAND
FF57 20
FF58 44
FF59 49
FF5A 46
FF5B 20
FF5C 49
FF5D 53
FF5E 20
FF5F 04 FCB 4
FF60 0D MSGS0 FCB CR,LF,0
FF61 0A
FF62 00
FF63 53 FCC 'S00600004844521B'
FF64 30
FF65 30
FF66 36
FF67 30
FF68 30
FF69 30
FF6A 30
FF6B 34
FF6C 38
FF6D 34
FF6E 34
FF6F 35
FF70 32
FF71 31
FF72 42
FF73 04 FCB 4
FF74 0D MSGS1 FCB CR,LF,0,0,'S,'1,4
FF75 0A
FF76 00
FF77 00
FF78 53
FF79 31
FF7A 04
FF7B 0D MSGS9 FCB CR,LF,0
FF7C 0A
FF7D 00
FF7E 53 FCC 'S9030000FC'
FF7F 39
FF80 30
FF81 33
FF82 30
FF83 30
FF84 30
FF85 30
FF86 46
FF87 43

	*			DELIMITER
FFD1 7E FA89	*	JMP	TSTEOL	TEST CHAR IN ACCA FOR END-OF-LINE
FFD4 7E F9C7	*	JMP	COMAND	SEARCH SPECIFIED COMMAND LIST FOR A COMMAND
FFD7 7E FA2E	*	JMP	TYPCMD	TYPES OUT COMMAND NUMBER "COMNUM" IN LIST ACCA
FFDA 7E FBFD	*	JMP	OUT1BY	DISPLAY THE 1 BYTE NUMBER POINTED AT BY IX
FFDD 7E FC04	*	JMP	OUT2BY	DISPLAY THE 2 BYTE NUMBER POINTED AT BY IX
FFE0 7E FD8C	*	JMP	GETLIN	GET A LINE OF INPUT INTO THE TTY BUFFER
FFE3 7E FE4B		JMP	OUTSTR	OUTPUT CHAR STRING IX POINTS TO
FFE6 7E FEC7	*	JMP	DOCRLF	SEND CR-LF WITH DELAY & ZERO LINE COUNT
FFE9 7E FE76	*	JMP	OUTCHR	LIKE TOACIA, BUT WITH FOLDING, CR DELAY, & LF INSERTION
FFEC 7E FEE5		JMP	TOACIA	SEND ACCA TO ACIA1
FFEF 7E FE59	*	JMP	INPCHR	GET A CHAR FROM AN ACIA & RETURN IT IN ACCA
FFF2 7E F425		JMP	PROMPT	TO PROMPT FOR NEW COMMAND
FFF5 7E F400		JMP	START	START OF MONDEB

	*			* INTERRUPT VECTORS
FFF8 FF99	FFF8	FDB	INTADR	REGULAR INTERRUPT
FFFA FFA6		FDB	SWIADR	SOFTWARE INTERRUPT
FFFC FF9E		FDB	NMIADR	NON-MASKABLE INTERRUPT
FFFE FFA3		FDB	RESADR	RESET INTERRUPT

* ======
* VARIABLES FOR INTER-ROUTINE COMMUNICATION

7000		ORG	\$7000	
7000 0002	INTVEC	RMB	2	INTERRUPT ADDRESS POINTER
7002 0002	NMIVEC	RMB	2	NON-MASKABLE INTERRUPT ADDRESS POINTER
7004 0002	SWIVEC	RMB	2	SOFTWARE INTERRUPT ADDRESS POINTER
	*			
7006 0002	SP	RMB	2	SAVED STACK POINTER
7008 0002	COMADR	RMB	2	ADDRESS OF BEGINNING OF COMMAND LISTS FOR COMAND
700A 0002	SYNPTR	RMB	2	INPUT LINE CHARACTER POINTER FOR GOOD SYNTAX
700C 0002	LINPTR	RMB	2	INPUT LINE CHARACTER POINTER
	*			(CONTENT = OR > CONTENT OF SYN PTR)
700E 0001	BOLFLG	RMB	1	"BEGINNING OF LINE" FLAG
700F 0001	DELIM	RMB	1	CHARACTER(S) PERMITTED AS VALID COMMAND/MODIFIER DELIMITER

7010 0001	IBCODE RMB	1	INPUT BASE (1=HEX, 2=DEC, 3=OCT)
7011 0001	DBCODE RMB	1	DISPLAY BASE (1=HEX, 2=DEC, 3=OCT, 4=BIN)
7012 0001	DBNBR RMB	1	DISPLAY BASE NUMBER (E.G., 16,10,8, OR 2)
7013 0001	NBRHI RMB	1	MOST SIGNIFICANT BYTE OF SCANNED NUMBER
7014 0001	NBRLO RMB	1	LEAST SIGNIFICANT BYTE OF SCANNED NUMBER
7015 0002	RANGLO RMB	2	RANGE LOWER LIMIT PICKED UP BY GTRANG
7017 0002	RANGHI RMB	2	RANGE UPPER LIMIT PICKED UP BY GTRANG
7019 0002	LASTGO RMB	2	LAST SPECIFIED GOTO ADDRESS
701B 0002	VERFRM RMB	2	BEGINNING ADDRESS OF RANGE TO CHECKSUM VERIFY
701D 0002	VERTO RMB	2	ENDING ADDRESS OF RANGE TO CHECKSUM VERIFY
701F 0001	CHKSUM RMB	1	CHECKSUM OF RANGE GIVEN IN THE VERIFY COMMAND
7020 0002	BRKADR RMB	2	ADDRESS OF INSERTED BREAKPOINT
7022 0001	BRKINS RMB	1	INSTRUCTION WHICH SHOULD BE THERE NORMALLY
7023 0001	INPFLG RMB	1	ALTERNATE INPUT DESTINATION FLAG
7024 0002	INPADR RMB	2	ALTERNATE ADDRESS THAT THE * INPUT CHARACTERS ARE TO COME FR * OM
7026 0001	OUTFLG RMB	1	ALTERNATE OUTPUT DESTINATION FLAG
7027 0002	OUTADR RMB	2	ALTERNATE ADDRESS THAT THE * OUTPUT CHARACTERS ARE TO GO TO
7029 0001	HDXFLG RMB	1	HALF-DUPLEX TERMINAL FLAG (IF NON-ZERO, NO ECHO)
702A 0001	CPLCNT RMB	1	"CHARACTERS PER LINE" COUNT
702B 0001	CPLMAX RMB	1	"CHARACTERS PER LINE" MAXIMUM
702C 0002	BUFBEG RMB	2	INPUT LINE START OF BUFFER
702E 0002	BUFEND RMB	2	INPUT LINE END OF BUFFER
7030 0048	TTYBUF RMB	72	START OF INPUT LINE BUFFER
7078 0001	TTYEND RMB	1	END OF INPUT LINE BUFFER
7079 0038		56	MAIN STACK STORAGE

70B1 0007	STACK	RMB	7	STACK STORAGE FOR RTI INSTRUCTION
* TEMPORARY (LOCALLY USED) VARIABLES				
70B8 0002	TEMP1	RMB	2	IN: MAIN
70BA 0002	TEMP2	RMB	2	IN: MAIN
70BC 0002	TEMP3	RMB	2	IN: FNDSTR,MAIN
70BE 0002	TEMP4	RMB	2	IN: MAIN
70C0 0002	TEMP5	RMB	2	IN: MAIN
70C2 0002	TEMP6	RMB	2	IN: MAIN
70C4 0002	TEMP7	RMB	2	IN: MAIN
70C6 0002	TEMP8	RMB	2	IN: MAIN
70C8 0002	TEMP9	RMB	2	IN: MAIN
70CA 0002	TEMP10	RMB	2	IN: MAIN
70CC 0002	TEMP11	RMB	2	IN: MAIN
*XTEMP IS NOT TO BE USED TO SAVE IX BETWEEN ROUTINES				
70CE 0002	XTEMP	RMB	2	USED BY DUMP, TYPCMD, OUTNUM
70D0 0002	XTEMP1	RMB	2	USED BY OUTNUM
70D2 0002	XTEMP2	RMB	2	USED BY NUMBER
70D4 0001	NUMMAT	RMB	1	USED IN COMMAND
70D5 0001	LISNUM	RMB	1	USED ON COMMAND
70D6 0001	COMNUM	RMB	1	USED IN COMMAND
70D7 0002	LISPTR	RMB	2	USED IN COMMAND
70D9 0001	DECDIG	RMB	1	DECIMAL DIGIT BEING BUILT (DECIMAL OUTPUT BASE)
70DA 0001	NUMBH1	RMB	1	USED BY OUTNUM
70DB 0001	NUMBLO	RMB	1	USED BY OUTNUM
70DC 0002	NBR2X	RMB	2	USED BY NUMBER
70DE 0002	TIMCON	RMB	2	DELAY TIME CONSTANT
70E0 0001	BYTECT	RMB	1	RECORD BYTE COUNT USED IN LOAD COMMAND
70E1 0001	CKSM	RMB	1	RECORD CHECKSUM USED IN LOAD COMMAND
*CONVENIENT EQUIVALENCES FOR LOCAL VARIABLES				
70B8	MEMADR	EQU	TEMP1	DISPLAY, SET, SEARCH, TEST
70BA	STRNUM	EQU	TEMP2	FNDSTR
70BB	EOSCHR	EQU	TEMP2+1	FNDSTR
*FOR "SEARCH" COMMAND				
70BA	BYTPTR	EQU	TEMP2	
70BC	NBYTES	EQU	TEMP3	
70BD	NBRMAT	EQU	TEMP3+1	
70BE	BYTSTR	EQU	TEMP4	

END

SYMBOL TABLE

ACIA1	7F43	DISPL2	F637	INPCH3	FE6A	NUMLUP	FB60	SET3	F697
ACIA2	7F45	DISPL3	F644	INPCHR	FE59	NUMMAT	70D4	SET4	F6A9
BADS1	F45A	DISPL4	F64D	INPFLG	7023	NXTCHR	FD90	SET5	F6C6
BADS2	F465	DISPL5	F64F	INT	F844	OCTNUM	FBBC	SET6	F6E4
BADSYN	F457	DISPL6	F654	INTADR	FF99	OUT1BY	FBFD	SET7	F6EC
BAKSLA	005C	DISPL7	F65F	INTVEC	7000	OUT2	F4E3	SET8	F6F4
BLDADR	F978	DISPL8	F662	ISDLM2	FAA8	OUT2A4	F4F2	SET9	F701
BOLFLG	700E	DISPL9	F66D	ISDLM3	FAB2	OUT2BY	FC04	SHFTER	FBED
BREAK	F558	DISPLA	F608	ISDLM4	FABC	OUT4	F4EA	SHIFT	FBE4
BREAK1	F56D	DISREG	F4C7	JMP256	F400	OUTADR	7027	SHIFT2	FBDF
BREAK2	F57E	DLMNO	FAD3	JMPCMD	F476	OUTBAK	FDE2	SKPDL1	FA6F
BREAK3	F58E	DLMYES	FAD0	JMPHI	00F4	OUTBIN	FC9F	SKPDL2	FA7A
BREAK4	F5A6	DOCRL1	FED9	JMPLO	0085	OUTCH1	FE8A	SKPDL3	FA81
BREAK5	F5B2	DOCRLF	FEC7	JMPTBL	F485	OUTCH2	FE92	SKPDLM	FA69
BREAK6	F5B5	DUMP	F885	LASTGO	7019	OUTCH3	FE98	SP	7006
BRKADR	7020	DUMP1	F88B	LASWAS	FE02	OUTCH4	FE9D	STACK	70B1
BRKINS	7022	DUMP10	F910	LEFSHF	FCB8	OUTCH5	FEA3	START	F400
BUFBEG	702C	DUMP2	F892	LF	000A	OUTCH6	FEAB	STORIT	FDD8
BUFEND	702E	DUMP3	F89A	LINPTR	700C	OUTCH7	FEBE	STRNUM	70BA
BYTECT	70E0	DUMP4	F8A5	LISNUM	70D5	OUTCH8	FEC0	SUMNUM	FAD6
BYTPTR	70BA	DUMP5	F8AD	LISPTR	70D7	OUTCHR	FE76	SWI	F854
BYTSTR	70BE	DUMP6	F8B5	LOAD	F924	OUTDE1	FC4B	SWIADR	FFA6
C100	FC7B	DUMP7	F8C7	LOAD1	F936	OUTDE2	FC52	SWIVEC	7004
C10K	FC77	DUMP8	F8C9	LOAD2	F955	OUTDE3	FC55	SYNPTR	700A
CHKSUM	701F	DUMP9	F8E9	LOAD3	F961	OUTDE4	FC60	TEMP1	70B8
CKSM	70E1	ECHO	FDE9	LOAD4	F972	OUTDE5	FC72	TEMP10	70CA
CKSUM	F757	ECHO1	FDF1	LSH2	FCB3	OUTDEC	FC3D	TEMP11	70CC
CKSUM1	F75C	EOSCHR	70BB	MATCH	FA07	OUTEND	FE57	TEMP2	70BA
CLI	F529	ERROR	FAD5	MEMADR	70B8	OUTEQ	FBF7	TEMP3	70BC
CMD3	F9E3	FFF8	FFF8	MFAIL	FA2B	OUTFLG	7026	TEMP4	70BE
CMD4	F9EC	FNDST1	FA58	MSGBAT	FF18	OUTHEX	FC27	TEMP5	70C0
COMADR	7008	FNDST2	FA5E	MSGCCL	FF32	OUTIT	FCAD	TEMP6	70C2
COMAND	F9C7	FNDST3	FA68	MSGCNH	FF8B	OUTNUM	FC0B	TEMP7	70C4
COLMLST	FCD7	FNDSTR	FA51	MSGCSO	FF3D	OUTOC1	FC8D	TEMP8	70C6
COMMUN	70D6	GETCHR	FCC0	MSGDIS	FF56	OUTOCT	FC81	TEMP9	70C8
COMPAT	F878	GETCMD	F44E	MSGHED	FEF2	OUTP2	F91D	TEST	F7ED
COMPAR	F85C	GETIT	FD9E	MSGLTL	FF07	OUTSD	F87B	TEST1	F7F8
CONERR	FBD9	GETLIN	FD8C	MSGNBR	FF10	OUTSP	FBF1	TEST2	F806
CONTIN	F604	GETLST	FCCD	MSGNVE	FF22	OUTST1	FE4C	TEST3	F813
COPY	F52C	GOTO	F514	MSGPRM	FEFF	OUTSTR	FE4B	TEST4	F824
COPY1	F539	GOTO1	F521	MSGSO	FF60	PROMP1	F434	TIMCON	70DE
COPY2	F552	GTRAN1	FB03	MSGS1	FF74	PROMPT	F425	TIMDE1	F9C0
COPY3	F555	GTRAN2	FB0B	MSGS9	FF7B	RANGHI	7017	TIMDEL	F9BD
CPLCNT	702A	GTRAN3	FB18	MSGSIS	FF4E	RANGLO	7015	TOACI1	FEE8
CPLMAX	702B	GTRAN4	FB20	MSGSWI	FF01	RDBYTE	F986	TOACIA	FEE5
CR	000D	GTRAN5	FB26	MSGVER	FF1F	REG	F4C7	TSTCMA	FAAC
DBASE	F5CE	GTRAN6	FB27	NBR2X	70DC	RESADR	FFA3	TSTCR	FDAB
DBASE1	F5D9	GTRAN7	FB2D	NBRHI	7013	RUBNOW	FDF3	TSTCR1	FDB3
DBASE2	F5DF	GTRAN8	FB30	NBRLO	7014	SEAR10	F7EA	TSTCR2	FDBE
DBASE3	F5EE	GTRANG	FAFC	NBRMAT	70BD	SEARC1	F774	TSTDLM	FA94
DBASE4	F5F2	GUDNUM	FBCD	NBYTES	70BC	SEARC2	F793	TSTE01	FA93
DBCODE	7011	HDXFLG	7029	NEXCOM	FA1B	SEARC3	F79F	TSTEOL	FA89
DBNBR	7012	HEXN1	FB89	NMATCH	FA11	SEARC4	F7AB	TTYBUF	7030
DBTBL	F5EA	HEXNUM	FB7B	NMI	F84C	SEARC5	F7B8	TTYEND	7078
DECDIG	70D9	IBASE	F5B8	NMIADR	FF9E	SEARC6	F7BB	TYPCM1	FA40

DECNUM	FB95	I BASE1	F5C3	NMIVEC	7002	SEARC7	F7CF	TYPCM2	FA4C
DELAY	F9B5	I BASE2	F5C8	NOMORE	F46F	SEARC8	F7DC	TYPCM2	FA2E
DELETE	007F	I BCODE	7010	NOTCTC	FDCA	SEARC9	F7E7	TYPSWI	F510
DELIM	700F	I NHEX	F998	NOTEOL	FDBF	SEARCH	F766	TYPSWI	F501
DIFNUM	FAE9	I NHEX1	F9AD	NULLS	F913	SEI	F526	VERFRM	701B
DISNU1	FC28	I NHEX2	F9AE	NULLS1	F916	SET	F673	VERIF1	F740
DISNU2	FC36	I NILST	F9D1	NUMBER	FB47	SET1	F68A	VERIF2	F74F
DISNU3	FC92	I INITAL	FE08	NUMBHI	70DA	SET10	F70E	VERIFY	F720
DISNU4	FCA2	I NPADR	7024	NUMBLO	70DB	SET11	F71A	VERTO	701D
DISP10	F670	I NPCH1	FE66	NUMIN1	FB43	SET12	F71D	XTEMP	70CE
DISPL1	F620	I NPCH2	FE69	NUMINX	FB3C	SET2	F694	XTEMP1	70D0

type cross.ref

CROSS REFERENCE TABLE

ACIA1	00600*	80040	80200	81800	86160	86280			
ACIA2	00640*	80080	80240						
BADS1	03480*	03640							
BADS2	03520	03760*							
BADSYN	03400*	04160	09960	12200	17160	21320	27360	35240	55760
BAKSLA	75600*	78240	79000						
BLDADR	37120	38080*							
BOLFLG	01920	47960	70360	90240*					
BREAK	05440	10160*							
BREAK1	10480	10760*							
BREAK2	10240	11120*							
BREAK3	10200	11480*							
BREAK4	11720	11960*							
BREAK5	11000	11240	11360	11880	12160*	12640	13600	14360	
BREAK6	11560	12200*	14160						
BRKADR	10360	10800	11120	11600	12040	91080*			
BRKINS	10600	10880	11280	91120*					
BUFBEG	01400	02600	03400	19040	75840	78680	91600*		
BUFEND	01560	75960	91640*						
BYTECT	37040	37280	92920*						
BYTPTR	24160	24840	24960	93280*					
BYTSTR	24120	25600	93400*						
C100	66120	67560*							
C10K	65880	67200	67480*						
CHKSUM	21920	21960	22200	91000*					
CKSM	36880	37560	38800	38840	92960*				
CKSUM	21880	22160	22800*	88600					
CKSUM1	22920*	23040							
CLI	05360	09040*							
CMD3	43160*	45040							
CMD4	43360*	43960							
COMADR	42520	80480	90080*						
COMAND	03080	11520	12440	13120	14080	15120	19280	32080	36040
	42160*	88920							
COMLST	45600	71160*	80440						
COMMUN	06440	07680	14280	15280	15480	16000	43000	44120	45000
	45760	92640*							
COMPAL	30080	30320	30560	31320*					
COMPAR	05920	30720*							
CONERR	58080	58920	59000	59640	60280	60400	60760	61600*	62400
CONTIN	05560	14560*							
COPY	05400	09240*							

COPY1	09480*	09880									
COPY2	09280	09360	09960*								
COPY3	08880	09080	09760	10000*							
CPLCNT	84120	84640	85360	85760	91520*						
CPLMAX	79760	84160	84360	91560*							
CR	00760*	43640	44920	45800	46000	49000	71240	71320	71400		
	71480	71560	71640	71720	71800	71880	71960	72040	72120		
	72200	72280	72360	72440	72520	72600	72680	72760	72840		
	73040	73240	73320	73400	73480	73680	73880	73960	74040		
	74120	74200	74280	74480	74560	74760	83880	85040	86520		
	86600	87440	87560	87600	87680	87760					
DBASE	05520	13080*									
DBASE1	13200	13280*									
DBASE2	13400*	13480									
DBASE3	13160	13960*									
DBASE4	12840	14040*									
DBCODE	13280	13960	64480	79520	90440*						
DBNBR	13560	15880	79640	90480*							
DBTBL	13360	13720*									
DECDIG	66280	66560	66920	92720*							
DECNUM	58400	59600*									
DELAY	06040	39760*									
DELETE	75640*	77440	77960	78800							
DELIM	01720	49920	90320*								
DIFNUM	31160	52760*									
DISNU1	65080*	65520									
DISNU2	65320	65440*									
DISNU3	68080	68320*	68600								
DISNU4	68920*	69120									
DISP10	16960	17200*	17960								
DISPL1	15440*	17080									
DISPL2	15680	15960*									
DISPL3	16040	16320*									
DISPL4	16400	16520*									
DISPL5	16480	16560*									
DISPL6	15520	16680*									
DISPL7	16200	16840*									
DISPL8	16600	16920*									
DISPL9	14800	15160	17160*	17440	17800	18280					
DISPLA	05600	14760*									
DISREG	06320*	08280									
DLMNO	50080	50320	50920	51160	51480*						
DLMYES	49840	50120	50360	50560	50840	51080	51320*				
DOCRL1	85600*	85720									
DOCRLF	01120	01880	03840	15760	29800	76560	77000	83960	84600		
	84960*	89160									
DUMP	05960	31840*									
DUMP1	32040*	32280	32480								
DUMP10	32160	35240*	36080								
DUMP2	32160*										
DUMP3	32240	32360*									
DUMP4	32120	32560*									
DUMP5	32600	32680*									
DUMP6	33040*	34920									
DUMP7	33280	33440*									
DUMP8	33400	33560*									
DUMP9	34360*	34440									
ECHO	78080	78360*	79200								
ECHO1	78400	78480*									
EOSCHR	46840	47240	93160*								
ERROR	50720	51640*									
FFF8	89480*										
FNDST1	46960*	47280									

FNDST2	47160*	47320								
FNDST3	47040	47400*								
FNDSTR	42720	45680	45840	46800*						
GETCHR	25920	26560	43360	48680	57760	70200*	88640			
GETCMD	02080	02960*								
GETIT	76000	76320*								
GETLIN	02320	19000	75840*	89080						
GETLST	25840	26520	43520	44800	70680*	88680				
GOTO	05280	08400*								
GOTOL	08440	08640*								
GTRAN1	53680	53880*								
GTRAN2	53720	54120*								
GTRAN3	53960	54480*								
GTRAN4	54240	54640*								
GTRAN5	54360	54880*	55000							
GTRAN6	54680	54960*								
GTRAN7	54560	55200*								
GTRAN8	54320	54960	55360*							
GTRANG	09240	14760	17360	21520	23960	27600	31840	53640*	88720	
GUDNUM	57920	61280*								
HDXFLG	76920	78360	79880	91440*						
HEXN1	58840	59160*								
HEXNUM	58280	58800*								
IBASE	05480	12400*								
IBASE1	12520	12600*								
IBASE2	12480	12760*								
IBCODE	12600	12760	58200	79480	90400*					
INHEX	38360	38640	39000*							
INHEX1	39160	39400*								
INHEX2	39080	39240	39320	39520*						
INILST	42280	42520*								
INITAL	01000	79440*								
INPADR	36240	81920	91240*							
INPCH1	81720	81920*								
INPCH2	81840	81960*								
INPCH3	82000*	82080								
INPCHR	36400	36600	39000	76320	81640*	89280				
INPFLG	36280	37920	79800	81680	91200*					
INT	05800	30000*								
INTADR	87920*	89480								
INTVEC	30040	87920	89840*							
ISDLM2	50000	50200*								
ISDLM3	50240	50440*								
ISDLM4	50480	50680*								
JMP256	05120*	05160								
JMPCMD	03240	04320*								
JMPHI	04480	05080*	05120							
JMPL0	04520	05160*								
JMPTBL	05000*	05080	05160							
LASTGO	08520	08640	90800*							
LASWAS	78840	79120*								
LEFSHF	68160	68360	68920	69560	69600	69800*				
LF	00800*	18800	42680	43560	44840	45640	49080	72880	73080	
		73520	73720	74320	74600	74800	83680	85120	87440	87560
			87600	87680						
LINPTR	03480	25480	26920	43200	44160	54120	57400	61280	70200	
		70320	90200*							
LISNUM	42160	42640	92600*							
LISPTR	25640	42760	70680	70800	92680*					
LOAD	06000	36000*								
LOAD1	36120	36400*	36480	36800	37600					
LOAD2	37200*	37440								
LOAD3	37320	37560*								

OUTCH4	82600	83680*							
OUTCH5	83720	83880*							
OUTCH6	83920	84120*							
OUTCH7	84200	84600*							
OUTCH8	84400	84480	84640*						
OUTCHR	03800	16560	35440	46080	62600	62840	69360	77320	78280
	78440	79040	81160	82520*	89200				
OUTDE1	65760	66120*							
OUTDE2	66000	66280*	67240						
OUTDE3	66400*	66600							
OUTDE4	66480	66720*							
OUTDE5	65560	67280*	68640	69160					
OUTDEC	64680	65720*							
OUTEND	81120	81280*							
OUTEQ	07800	16760	29440	62800*					
OUTFLG	32640	35160	79840	82560	82840	91320*			
OUTHEX	64600	65000*							
OUTIT	65440	66960	68040	68200	68520	69040	69320*		
OUTNUM	63160	63520	64320*						
OUTOC1	67960	68160*							
OUTOCT	64760	67840*							
OUTP2	34000	34160	34200	34360	34720	35680*			
OUTSD	31080	31240	31440*						
OUTSP	03560	07640	15840	16160	16680	18960	27200	29600	62560*
OUTST1	81040*	81240							
OUTSTR	01200	02240	08000	11840	12000	22400	22600	29720	31440
	32880	33800	35080	37760	39560	76160	81000*	89120	
PROMPL	01760	02200*	03880						
PROMPT	01880*	02480	02840	03200	04080	89320			
RANGHI	09720	16920	17600	17920	21760	26080	26840	28920	31480
	33040	33160	34880	52080	52280	52840	53040	54520	90720*
RANGLO	09440	09680	09840	14920	17560	21680	25400	27760	30760
	33080	33200	34120	34320	34560	34800	52000	52200	52760
	52960	53920	55440	90680*					
RDBYTE	36960	37200	38080	38160	38360*				
REG	05240	06240*							
RESADR	88160*	89600							
RUBNOW	77480	78680*							
SEAR10	26120	26880	27400*	28960					
SEARC1	24360*	25000							
SEARC2	24400	25160*							
SEARC3	25600*	27000							
SEARC4	25920*	26160							
SEARC5	26040	26280*							
SEARC6	26320*	26640							
SEARC7	26760*	27280							
SEARC8	26440	27120*							
SEARC9	24000	24440	24720	25200	27360*	27640			
SEARCH	05720	23960*							
SEI	05320	08840*							
SET	05640	17360*							
SET1	17880*	18040							
SET10	20800	21080*							
SET11	19320	19520	21120	21320*	21600				
SET12	18840	19360	21360*	22040	22440	22640			
SET2	17640	18160*	18600						
SET3	18200*	19120							
SET4	18240	18720*							
SET5	17400	19240*	19840	20080	20320	20640	20960	21240	
SET6	19760	19960*							
SET7	20000	20200*							
SET8	20240	20440*							
SET9	20480	20760*							

SHFTER	62160	62320*						
SHIFT	59840	60920	61880	61920	62080*			
SHIFT2	59160	59200	60040	60880	61880*			
SKPDLL1	48120*	48760						
SKPDLL2	48000	48240	48440*					
SKPDLL3	48520	48680*						
SKPDLM	04040	42240	47920*	57520	88800			
SP	00960	06320	06880	08080	14560	19560	21200	88240
STACK	00920	91800*						900000*
START	00880*	88160	88440	89360				
STORIT	77640	77720	77840*					
STRNUM	46800	47000	93120*					
SUMNUM	31000	52000*	55080					
SWI	05880	30480*						
SWIADR	88240*	89520						
SWIVEC	30520	80360	88280	89920*				
SYNPTR	01960	02640	18720	19080	43160	44200	48120	48720
	57360	61320	90160*					55360
TEMP1	37800	38120	38200	38240	91920*	93080		
TEMP10	92280*							
TEMP11	92320*							
TEMP2	91960*	93120	93160	93280				
TEMP3	29280	29680	33600	33960	92000*	93320	93360	
TEMP4	33680	34400	92040*	93400				
TEMP5	31920	32440	32560	92080*				
TEMP6	92120*							
TEMP7	92160*							
TEMP8	92200*							
TEMP9	92240*							
TEST	05760	27600*						
TEST1	27920*	29160						
TEST2	28120	28360*						
TEST3	28480	28720*	29840					
TEST4	28280	28640	29280*					
TIMCON	40160	80600	92880*					
TIMDEL1	40240*	40280						
TIMDEL	39800	40160*	40400	80720	88560			
TOACIL1	86160*	86200						
TOACIA	84680	85080	85160	85640	86080*	89240		
TSTCMA	50280*	50600						
TSTCR	76520	76680*						
TSTCR1	76720	76840*						
TSTCR2	76960	77040*						
TSTDLM	43400	44600	48480	49680*	57880	88840		
TSTEOL1	49040	49120	49200*					
TSTEOL	02800	48200	49000*	49760	88880			
TTYBUF	01360	91680*						
TTYEND	01520	91720*						
TYPCM1	45920*	46120						
TYPCM2	46040	46200*						
TYPCMD	07760	14320	45520*	88960				
TYPSW1	08160	08240*						
TYPSWI	07960*	80320						
VERFRM	21720	22840	90880*					
VERIF1	21560	22160*						
VERIF2	22240	22560*						
VERIFY	05680	21520*						
VERTO	21800	23000	90920*					
XTEMP	45520	46200	81640	82160	82720	83440	92440*	
XTEMP1	64320	67320	92480*					
XTEMP2	57160	57600	61360	61600	92520*			

END OF LISTING

Appendix C:

PAPERBYTE™ Bar Code Representation of MONDEB Object Code

Beginning on page 85 is a complete machine readable representation of the object code for *Mondeb*, as assembled in the listing found on pages 19 to 72 of this book.

This representation uses the absolute loader format, in which each bar code frame (one line of bars running from top to bottom of the page) contains a two byte address followed by data which is loaded in ascending order starting at that address.

The object code listing shown below gives the information in hexadecimal form, for use as a confirmation copy or for manual entry of this program.

For details on the frame format and absolute loader format used in this and all PAPERBYTE™ books, see the PAPERBYTE publication *Bar Code Loader* by Ken Budnick. This book contains a brief history on bar codes, a general bar code loader algorithm with flow charts and complete program listings for 6800, 6502 and 8080 or Z-80 based systems.

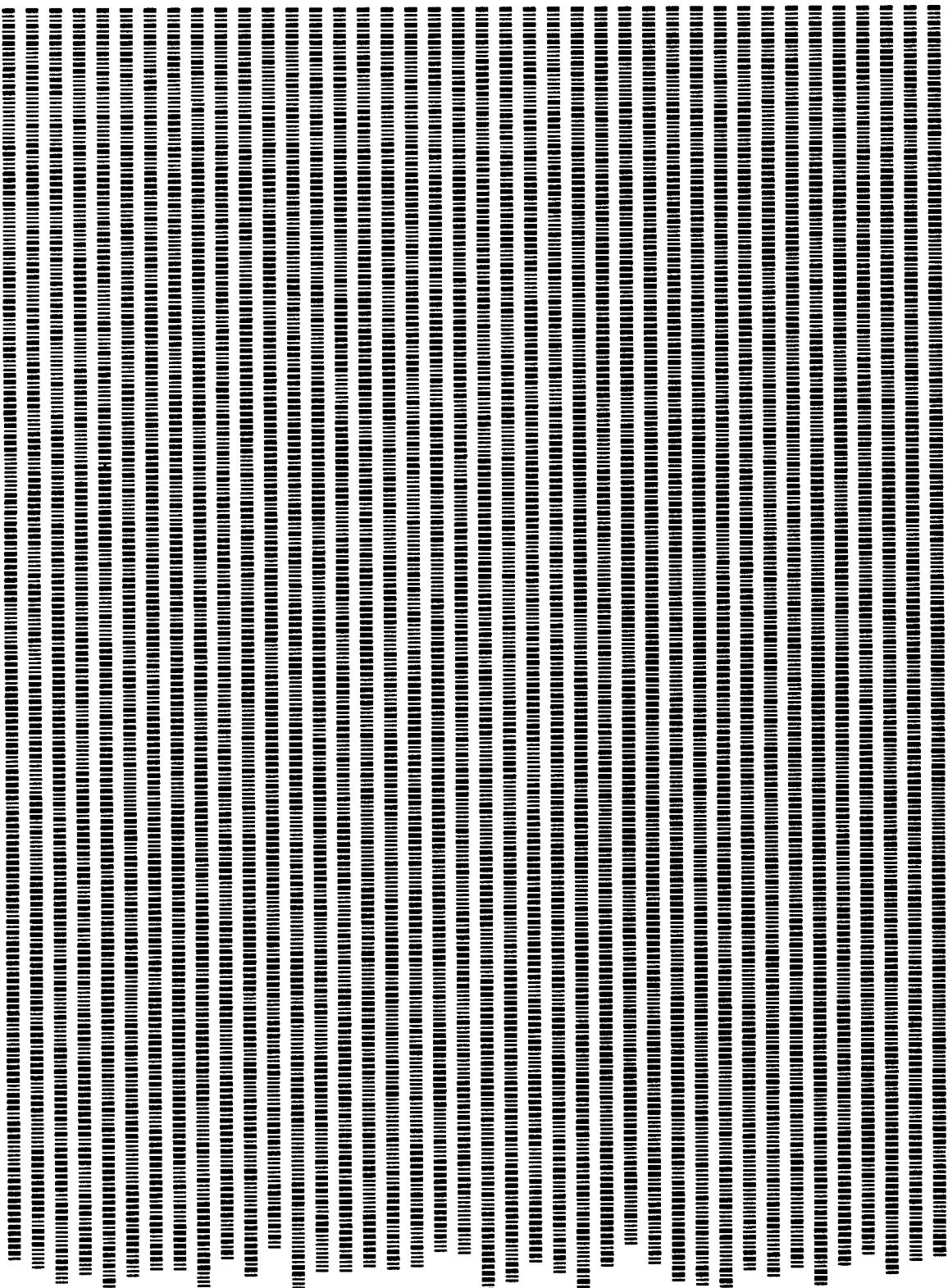
```
F400 8E 70 B1 BF 70 06 BD FE 08 BD FE C7 CE FE F2 BD
F410 FE 4B CE 70 2F FF 70 2C CE 70 78 FF 70 2E 86 03
F420 B7 70 0F 20 0F BD FE C7 7C 70 0E FE 70 0A A6 00
F430 81 3B 27 1A CE FE FF BD FE 4B BD FD 8C C1 03 27
F440 E4 FE 70 2C FF 70 0A A6 01 BD FA 89 27 D7 86 01
F450 BD F9 C7 27 D0 2E 1F FE 70 2C BC 70 0C 27 06 BD
F460 FB F1 08 20 F5 86 5E BD FE 76 BD FE C7 20 C5 BD
F470 FA 69 25 B1 20 E1 16 48 1B C6 F4 8B 85 C9 00 36
F480 37 30 EE 00 33 32 6E 00 7E F4 C7 7E F5 14 7E F5
F490 26 7E F5 29 7E F5 2C 7E F5 58 7E F5 B8 7E F5 CE
F4A0 7E F6 04 7E F6 08 7E F6 73 7E F7 20 7E F7 66 7E
F4B0 F7 ED 7E F8 44 7E F8 4C 7E F8 54 7E F8 5C 7E F8
F4C0 85 7E F9 24 7E F9 B5 FE 70 06 08 7F 70 D6 8D 13
F4D0 8D 11 8D 0F 8D 14 8D 12 8D 18 CE 70 06 BD FC 04
F4E0 7E F4 6F 8D 0D BD FB FD 08 39 8D 06 BD FC 04 08
F4F0 08 39 BD FB F1 7C 70 D6 86 05 BD FA 2E BD FB F7
F500 39 CE FF 01 BD FE 4B FE 70 06 6D 07 26 02 6A 06
F510 6A 07 20 B3 BD FB 47 27 08 FE 70 13 FF 70 19 6E
F520 00 FE 70 19 6E 00 0F 20 2C 0E 20 29 BD FA FC 2F
F530 21 BD FB 47 2F 1C FE 70 15 A6 00 FE 70 13 A7 00
F540 08 FF 70 13 FE 70 15 BC 70 17 27 09 08 FF 70 15
F550 20 E7 7E F4 57 7E F4 6F BD FB 47 2B 31 27 1F FE
F560 70 20 A6 00 81 3F 26 05 B6 70 22 A7 00 FE 70 13
F570 FF 70 20 A6 00 B7 70 22 86 3F A7 00 20 34 FE 70
F580 20 A6 00 81 3F 26 2B B6 70 22 A7 00 20 24 86 04
F590 BD F9 C7 2F 20 FE 70 20 A6 00 81 3F 27 08 CE FF
F5A0 10 BD FE 4B 20 0C CE FF 18 BD FE 4B CE 70 20 BD
F5B0 FC 04 7E F4 6F 7E F4 57 86 03 BD F9 C7 2B 09 2E
F5C0 02 86 01 B7 70 10 20 EA B6 70 10 36 20 24 86 03
F5D0 BD F9 C7 2B 19 2E 02 86 01 B7 70 11 CE F5 E9 08
F5E0 4A 26 FC A6 00 B7 70 12 20 C8 10 0A 08 02 B6 70
F5F0 11 36 86 04 BD F9 C7 33 2F BB 86 03 F7 70 D6 BD
F600 FA 2E 20 AE BE 70 06 3B BD FA FC 2F 60 FE 70 15
F610 FF 70 B8 86 06 BD F9 C7 2B 53 4A B7 70 D6 5F 5C
F620 CE 70 B8 7D 70 D6 2B 2C 5A 26 0C BD FE C7 BD FC
F630 04 BD FB F1 F6 70 12 FE 70 B8 7D 70 D6 2E 05 BD
F640 FB F1 20 1B A6 00 4D 26 04 86 2E 20 02 86 2B BD
F650 FE 76 20 0E BD FB F1 BD FC 04 BD FB F7 EE 00 BD
F660 FB FD BC 70 17 27 09 08 FF 70 B8 20 B3 7E F4 57
```

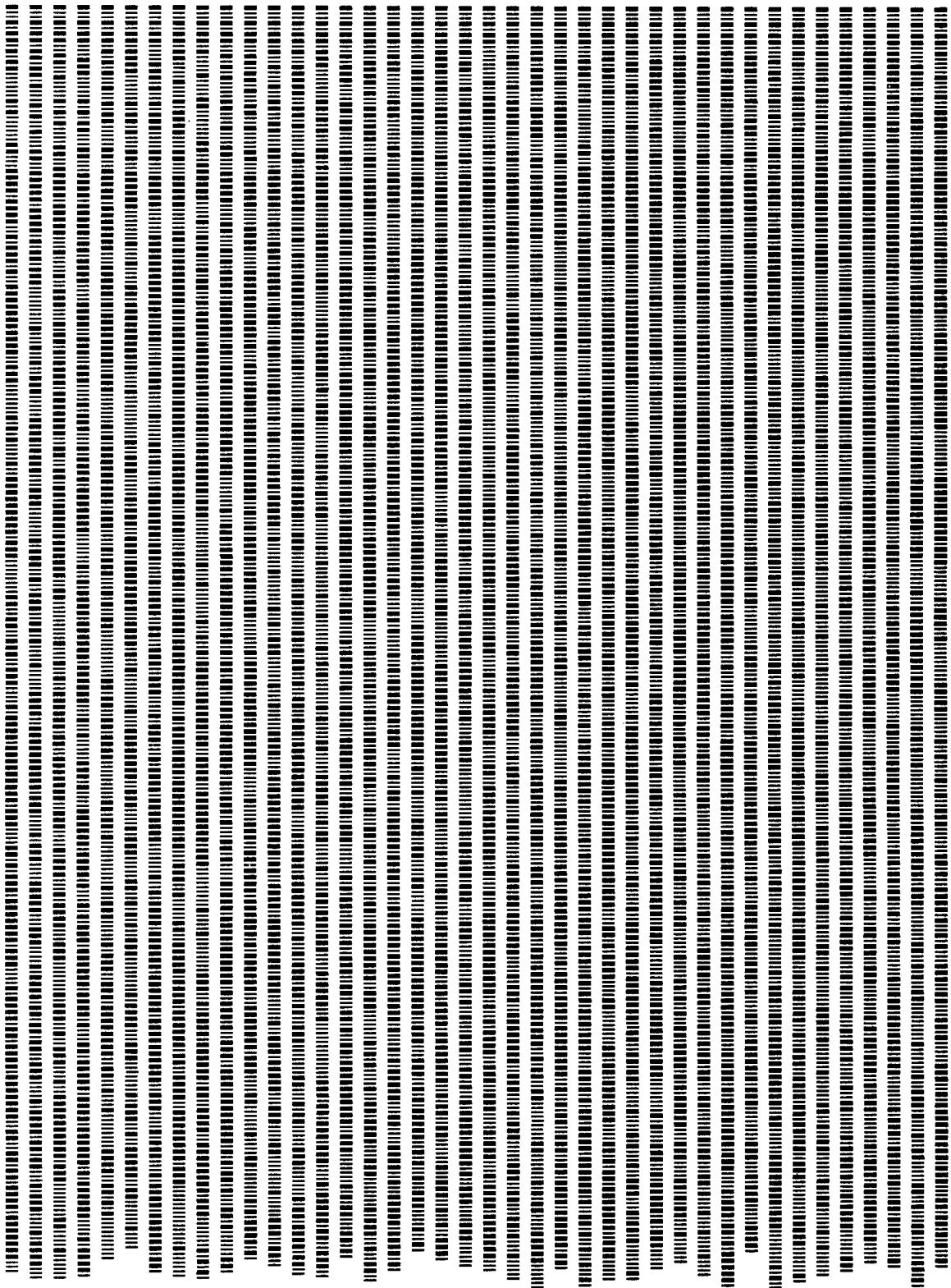
F670	7E	F4	6F	BD	FA	FC	2B	4E	27	F3	FE	70	15	BC	70	17
F680	27	12	BD	FB	47	2F	E6	B6	70	14	A7	00	BC	70	17	27
F690	DF	08	20	F6	FF	70	B8	BD	FB	47	27	0D	2D	CF	B6	70
F6A0	14	FE	70	B8	A7	00	08	20	EB	FE	70	0A	A6	00	81	0A
F6B0	26	6B	CE	70	B8	BD	FC	04	BD	FB	F1	BD	FD	8C	FE	70
F6C0	2C	FF	70	0A	20	D1	86	05	BD	F9	C7	2B	4D	27	4E	36
F6D0	BD	FB	47	32	2F	44	FE	70	06	F6	70	14	81	01	26	04
F6E0	E7	01	20	E2	81	02	26	04	E7	02	20	DA	81	03	26	04
F6F0	E7	03	20	D2	81	04	26	09	B6	70	13	A7	04	E7	05	20
F700	C5	81	05	26	09	B6	70	13	A7	06	E7	07	20	B8	81	06
F710	26	08	FE	70	13	FF	70	06	20	AC	7E	F4	57	7E	F4	6F
F720	BD	FA	FC	27	1B	2B	F3	FE	70	15	FF	70	1B	FE	70	17
F730	FF	70	1D	8D	22	B7	70	1F	CE	70	1F	BD	FB	FD	20	DD
F740	8D	15	B1	70	1F	26	08	CE	FF	1F	BD	FE	4B	20	CE	CE
F750	FF	22	BD	FE	4B	20	C6	4F	FE	70	1B	09	08	AB	00	BC
F760	70	1D	26	F8	43	39	BD	FA	FC	2F	7C	CE	70	BE	FF	70
F770	BA	7F	70	BC	BD	FB	47	27	1A	2D	6C	7C	70	BC	B6	70
F780	BC	81	06	2E	62	B6	70	14	FE	70	BA	A7	00	08	FF	70
F790	BA	20	E1	7D	70	BC	27	4F	FE	70	15	09	FF	70	0C	CE
F7A0	70	BD	FF	70	D7	7F	70	BD	BD	FC	CD	BD	FC	C0	11	27
F7B0	07	BC	70	17	27	34	20	F3	FF	70	B8	7C	70	BD	B6	70
F7C0	BD	B1	70	BC	27	16	BD	FC	CD	BD	FC	C0	11	27	EC	FE
F7D0	70	B8	BC	70	17	27	13	FF	70	0C	20	C3	CE	70	B8	BD
F7E0	FC	04	BD	FB	F1	20	E8	7E	F4	57	7E	F4	6F	BD	FA	FC
F7F0	2F	F5	FE	70	15	FF	70	B8	A6	00	36	6F	00	6D	00	27
F800	05	CE	FF	32	20	1E	6A	00	86	FF	A1	00	27	05	CE	FF
F810	3D	20	11	FE	70	B8	32	A7	00	BC	70	17	27	CC	08	FF
F820	70	B8	20	D4	FF	70	BC	CE	70	B8	BD	FC	04	BD	FB	F7
F830	FE	70	B8	BD	FB	FD	BD	FB	F1	FE	70	BC	BD	FE	4B	BD
F840	FE	C7	20	CF	BD	FB	3C	FF	70	00	20	2C	BD	FB	3C	FF
F850	70	02	20	24	BD	FB	3C	FF	70	04	20	1C	BD	FB	3C	FF
F860	70	15	BD	FB	3C	FF	70	13	BD	FA	D6	CE	FF	4E	8D	0B
F870	BD	FA	E9	CE	FF	56	8D	03	7E	F4	6F	BD	FE	4B	CE	70
F880	17	BD	FC	04	39	BD	FA	FC	7F	70	C0	86	02	BD	F9	C7
F890	27	13	2F	7C	81	01	27	02	20	F1	BD	FB	3C	FF	70	27
F8A0	7C	70	C0	20	E6	7D	70	C0	27	03	7C	70	26	8D	64	CE
F8B0	FF	60	BD	FE	4B	B6	70	18	B0	70	16	F6	70	17	F2	70
F8C0	15	26	04	81	10	25	02	86	0F	8B	04	B7	70	BC	80	03
F8D0	B7	70	BE	CE	FF	74	BD	FE	4B	5F	CE	70	BC	8D	3E	CE
F8E0	70	15	8D	39	8D	37	FE	70	15	8D	32	7A	70	BE	26	F9
F8F0	FF	70	15	53	37	30	8D	25	33	FE	70	15	09	BC	70	17
F900	26	B3	CE	FF	7B	BD	FE	4B	8D	09	7F	70	26	7E	F4	6F
F910	7E	F4	57	C6	1E	4F	BD	FE	76	5A	26	FA	39	EB	00	BD
F920	FB	FD	08	39	86	07	BD	F9	C7	2B	E5	27	09	BD	FB	3C
F930	FF	70	24	7C	70	23	BD	FE	59	81	53	26	F9	BD	FE	59
F940	81	39	27	2E	81	31	26	EE	7F	70	E1	BD	F9	86	80	02
F950	B7	70	E0	8D	23	8D	2F	7A	70	E0	27	05	A7	00	08	20
F960	F4	7C	70	E1	27	D0	CE	FF	22	BD	FE	4B	CE	70	B8	BD
F970	FC	04	7F	70	23	7E	F4	6F	8D	0C	B7	70	B8	8D	07	B7
F980	70	B9	FE	70	B8	39	8D	10	48	48	48	48	16	8D	09	1B
F990	16	FB	70	E1	F7	70	E1	39	BD	FE	59	80	30	2B	0F	81
F9A0	09	2F	0A	81	11	2B	07	81	16	2E	03	80	07	39	CE	FF
F9B0	8B	BD	FE	4B	39	BD	FB	3C	8D	03	7E	F4	6F	B6	70	DE
F9C0	4A	26	FD	09	26	F7	39	B7	70	D5	BD	FA	69	24	02	4F
F9D0	39	FE	70	08	B6	70	D5	C6	0A	8D	76	FF	70	D7	86	01
F9E0	B7	70	D6	FE	70	0A	FF	70	0C	7F	70	D4	BD	FC	C0	BD
F9F0	FA	94	26	13	BD	FC	CD	81	0A	27	16	81	0D	27	12	11

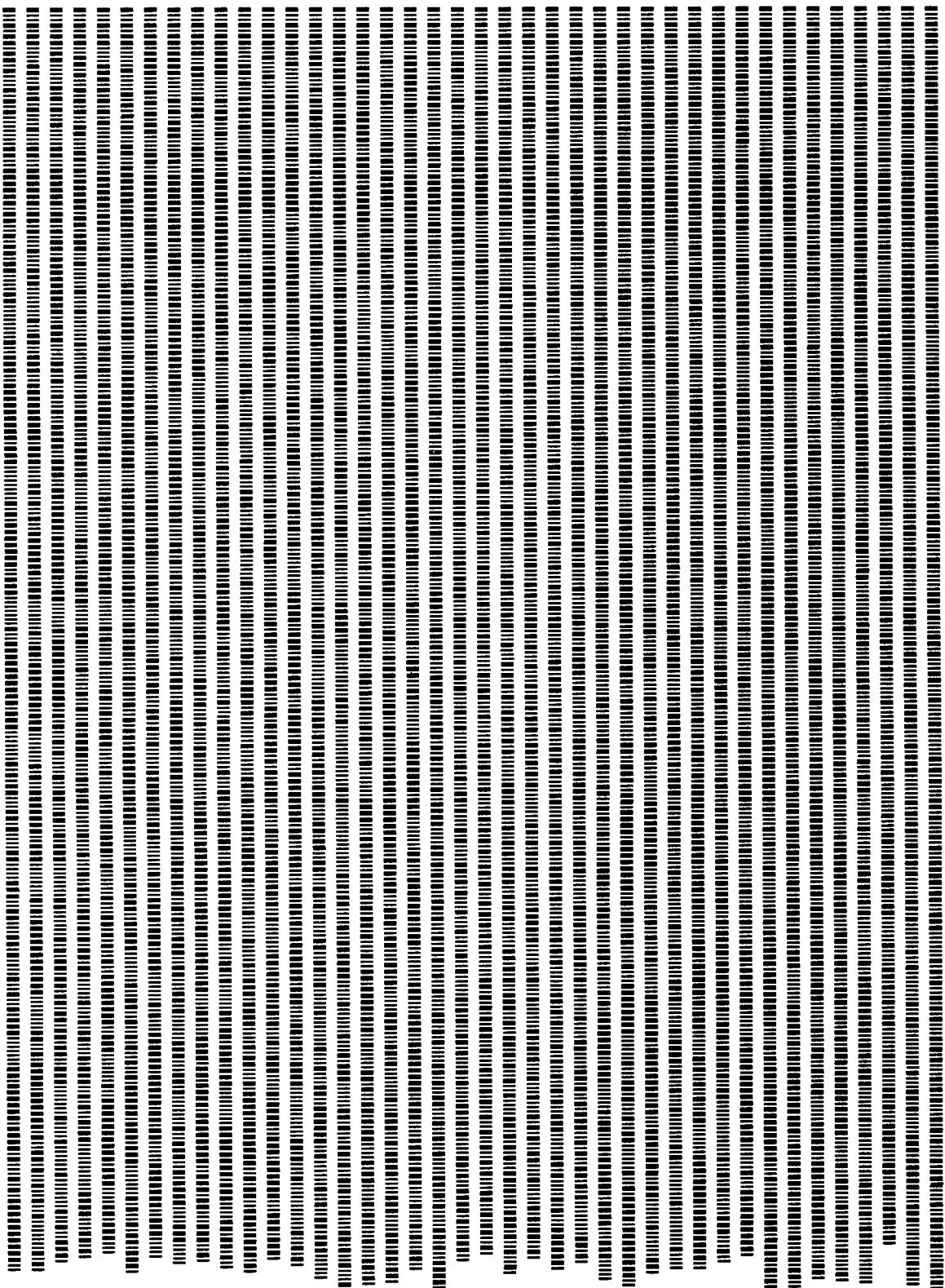
FA00	26	19	7C	70	D4	20	E5	B6	70	D6	FE	70	0C	FF	70	0A
FA10	39	7D	70	D4	27	05	BD	FA	94	26	EC	BD	FC	CD	81	0A
FA20	27	09	81	0D	26	F5	7C	70	D6	20	B8	4F	4A	39	FF	70
FA30	CE	37	CE	FC	D6	C6	0A	8D	18	B6	70	D6	C6	0D	8D	11
FA40	08	A6	00	81	0D	27	05	BD	FE	76	20	F4	FE	70	CE	33
FA50	39	B7	70	BA	F7	70	BB	5F	5C	F1	70	BA	27	0A	08	A6
FA60	00	B1	70	BB	27	F2	20	F6	39	0C	7D	70	0E	2E	0B	FE
FA70	70	0A	A6	00	8D	13	26	02	0D	39	E6	01	8D	16	26	01
FA80	39	BD	FC	C0	FF	70	0A	20	E6	81	0D	27	06	81	0A	27
FA90	02	81	3B	39	37	17	8D	F1	33	27	35	B6	70	0F	81	01
FAA0	26	06	CI	20	26	2D	20	28	81	02	26	06	CI	2C	26	23
FAB0	20	1E	81	03	26	06	C1	20	27	16	20	F0	81	04	26	15
FAC0	C1	30	2D	0C	C1	39	2F	0B	C1	41	2D	04	C1	5A	2F	03
FAD0	86	01	39	4F	39	3F	B6	70	16	BB	70	14	B7	70	18	B6
FAE0	70	15	B9	70	13	B7	70	17	39	B6	70	16	B0	70	14	B7
FAF0	70	18	B6	70	15	B2	70	13	B7	70	17	39	8D	49	2E	03
FB00	2D	09	39	FE	70	13	FF	70	15	20	0D	FE	70	0C	A6	00
FB10	81	3A	26	0C	8D	1A	2F	0E	FE	70	13	FF	70	17	20	0D
FB20	81	21	27	03	4F	4A	39	8D	07	2F	FB	8D	A9	86	01	39
FB30	FF	70	0A	FE	70	13	FF	70	15	8D	0C	39	8D	09	2E	03
FB40	7E	F4	57	FE	70	13	39	FF	70	D2	7F	70	13	7F	70	14
FB50	FE	70	0A	FF	70	0C	BD	FA	69	24	05	FE	70	D2	4F	39
FB60	BD	FC	C0	BD	FA	94	26	65	C0	30	2B	6D	B6	70	10	81
FB70	01	27	08	81	02	27	1E	81	03	27	41	C1	09	2F	0A	C1
FB80	11	2B	56	CI	16	2E	52	C0	07	8D	54	8D	52	FA	70	14
FB90	F7	70	14	20	CB	C1	09	2E	40	8D	49	FE	70	13	FF	70
FBA0	DC	8D	3C	4F	FB	70	DD	B9	70	DC	25	2D	FB	70	14	B9
FBB0	70	13	25	25	F7	70	14	B7	70	13	20	A4	C1	07	2E	19
FBC0	8D	1D	8D	20	FA	70	14	F7	70	14	7E	FB	60	FE	70	0C
FBDO	FF	70	0A	FE	70	D2	86	01	39	FE	70	D2	4F	4A	39	8D
FBE0	03	8D	01	39	78	70	14	79	70	13	25	01	39	31	31	20
FBFO	E8	86	20	BD	FE	76	39	86	3D	BD	FE	76	39	37	C6	01
FC00	8D	09	33	39	37	C6	02	8D	02	33	39	FF	70	D0	36	EE
FC10	00	FF	70	DA	B6	70	11	81	01	27	0C	81	02	27	1E	81
FC20	03	27	5E	81	04	27	78	58	BD	FC	B3	BD	FC	B3	84	0F
FC30	81	09	2F	02	8B	07	8D	75	5A	26	ED	20	35	5A	27	0B
FC40	CE	FC	77	B6	70	DA	F6	70	DB	20	07	CE	FC	7B	4F	F6
FC50	70	DA	7F	70	D9	E0	01	A2	00	25	05	7C	70	D9	20	F5
FC60	EB	01	A9	00	36	B6	70	D9	8D	43	32	08	08	8C	FC	81
FC70	26	E0	32	FE	70	D0	39	27	10	03	E8	00	64	00	0A	00
FC80	01	58	4F	C1	02	2E	06	8D	2A	8D	22	20	05	8D	29	8D
FC90	1C	5C	8D	1F	8D	22	84	07	8D	13	5A	26	F5	20	D3	58
FCA0	58	58	8D	14	84	01	8D	05	5A	26	F7	20	C5	8B	30	BD
FCB0	FE	76	39	8D	03	8D	01	39	78	70	DB	79	70	DA	49	39
FCC0	FE	70	0C	08	E6	00	FF	70	0C	7F	70	0E	39	FE	70	D7
FCDD0	08	A6	00	FF	70	D7	39	52	45	47	0D	47	4F	54	4F	0D
FCE0	53	45	49	0D	43	4C	49	0D	43	4F	50	59	0D	42	52	45
FCF0	41	4B	0D	49	42	41	53	45	0D	44	42	41	53	45	0D	43
FD00	4F	4E	54	49	4E	55	45	0D	44	49	53	50	4C	41	59	0D
FD10	53	45	54	0D	56	45	52	49	46	59	0D	53	45	41	52	43
FD20	48	0D	54	45	53	54	0D	49	4E	54	0D	4E	4D	49	0D	53
FD30	57	49	0D	43	4F	4D	50	41	52	45	0D	44	55	4D	50	0D
FD40	4C	4F	41	44	0D	44	45	4C	41	59	0D	0A	54	4F	0D	0A
FD50	48	45	58	0D	44	45	43	0D	4F	43	54	0D	42	49	4E	0D
FD60	0A	3F	0D	0A	2E	43	43	0D	2E	42	0D	2E	41	0D	2E	49
FD70	58	0D	2E	50	43	0D	2E	53	50	0D	0A	44	41	54	41	0D

FD80	55	53	45	44	0D	0A	46	52	4F	4D	0D	0A	FE	70	2C	5F
FD90	BC	70	2E	26	09	CE	FF	07	BD	FE	4B	C6	03	39	BD	FE
FDA0	59	84	7F	81	1A	26	04	BD	FE	C7	39	81	0D	27	04	81
FDB0	0A	26	0C	08	A7	00	7D	70	29	26	03	BD	FE	C7	39	81
FDC0	03	26	07	16	86	5E	BD	FE	76	39	81	7F	27	25	81	60
FDD0	23	06	81	7A	22	02	80	20	08	A7	00	C1	7F	27	03	16
FDE0	20	07	16	86	5C	BD	FE	76	17	7D	70	29	26	03	BD	FE
FDF0	76	20	9D	BC	70	2C	27	98	C1	7F	27	06	16	86	5C	BD
FE00	FE	76	A6	00	09	20	E2	0F	86	01	B7	70	10	B7	70	11
FE10	86	10	B7	70	12	86	48	B7	70	2B	7F	70	23	7F	70	26
FE20	7F	70	29	86	03	B7	7F	42	B7	7F	44	86	02	B7	7F	42
FE30	B7	7F	44	CE	F5	01	FF	70	04	CE	FC	D6	FF	70	08	86
FE40	53	B7	70	DE	CE	01	F4	BD	F9	BD	39	36	A6	00	81	04
FE50	27	05	8D	22	08	20	F5	32	39	FF	70	CE	7D	70	23	26
FE60	05	CE	7F	43	20	03	FE	70	24	09	A6	00	85	01	27	FA
FE70	A6	01	FE	70	CE	39	37	7D	70	26	27	21	FF	70	CE	FE
FE80	70	27	C6	02	F1	70	26	27	09	09	E5	00	27	FC	A7	01
FE90	20	06	A7	00	08	FF	70	27	FE	70	CE	33	39	81	0A	26
FEA0	02	33	39	81	0D	26	04	8D	1E	33	39	F6	70	2A	F1	70
FEB0	2B	2C	0B	CB	0A	F1	70	2B	2D	06	81	20	26	02	8D	07
FEC0	7C	70	2A	8D	20	33	39	36	37	86	0D	8D	18	86	0A	8D
FED0	14	F6	70	2A	54	54	54	54	5C	4F	8D	09	5A	26	FA	7F
FEE0	70	2A	33	32	39	36	86	02	B5	7F	42	27	FB	32	B7	7F
FEF0	43	39	4D	4F	4E	44	45	42	20	31	2E	30	30	0D	04	2A
FF00	04	0D	53	57	49	3A	04	54	4F	4F	20	4C	4F	4E	47	04
FF10	4E	4F	54	20	53	45	54	04	53	45	54	20	40	20	04	4F
FF20	4B	04	43	48	45	43	4B	53	55	4D	20	45	52	52	4F	52
FF30	20	04	43	41	4E	54	20	43	4C	45	41	52	04	43	41	4E
FF40	54	20	53	45	54	20	54	4F	20	4F	4E	45	53	04	53	55
FF50	4D	20	49	53	20	04	2C	20	44	49	46	20	49	53	20	04
FF60	0D	0A	00	53	30	30	36	30	30	30	30	34	38	34	34	35
FF70	32	31	42	04	0D	0A	00	00	53	31	04	0D	0A	00	53	39
FF80	30	33	30	30	30	30	46	43	0D	0A	04	43	48	41	52	20
FF90	4E	4F	54	20	48	45	58	0D	04	FE	70	00	6E	00	FE	70
FFA0	02	6E	00	7E	F4	00	BF	70	06	FE	70	04	6E	00		
FFB9	7E	F9	BD	7E	F7	57	7E									
FFC0	FC	C0	7E	FC	CD	7E	FA	FC	7E	FB	47	7E	FA	69	7E	FA
FFD0	94	7E	FA	89	7E	F9	C7	7E	FA	2E	7E	FB	FD	7E	FC	04
FFE0	7E	FD	8C	7E	FE	4B	7E	FE	C7	7E	FE	76	7E	FE	E5	7E
FFF0	FE	59	7E	F4	25	7E	F4	00	FF	99	FF	A6	FF	9E	FF	A3

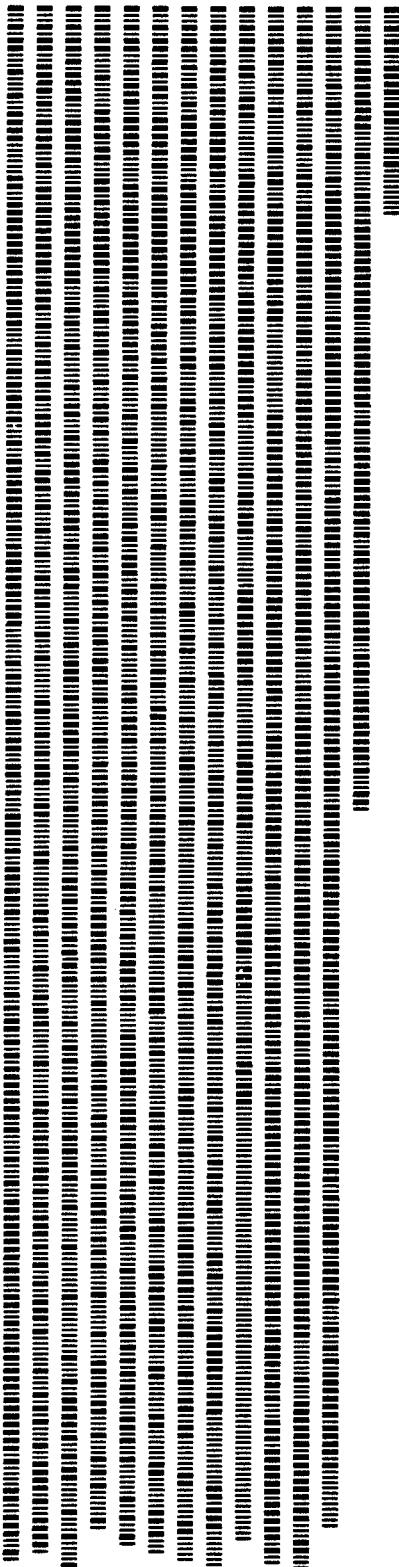
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
4	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	6	6	6	6	6	6	6	7
0	1	2	4	5	7	8	A	B	C	D	E	F	1	2	4	5	6	8	9	B	C	D	E	F	7	7	
0	6	D	4	B	2	A	0	D	D	4	B	C	1	9	0	7	E	6	E	9	C	2	3	1	8	0	







F	F	F	F	F	F	F	F	F	F	F	F	F	F
E	E	F	F	F	F	F	F	F	F	F	F	F	F
D	F	0	2	3	5	6	8	9	B	C	E	F	F
B	2	A	3	B	3	C	5	E	7	D	3		8



A Note About Bar Codes . . .

Bar codes are the newest form of machine readable data representation. They are used in all PAPERBYTE™ software products in BYTE magazine articles and self contained book publications and combine efficiency of space, low cost, and ease of data entry with the need for mass produced machine readable representations of software. Bar codes were originally used for product identification in inventory control and supermarket checkout applications. Today, because of their direct binary representation of data, they are an ideal computer compatible communications medium. In the application of bar codes to software distribution (such as PAPERBYTE books and articles), the use of a simple but reliable optical scanning wand and an appropriate program provides a convenient means for the user to acquire software.

Our intent in making PAPERBYTE software available in bar code form is to provide a method of conveying machine readable information from documentation to the memories and mass storage of a user's system on a one time basis. We suggest that the user of software obtained in this manner should locally record the data on the mass storage devices of his system after the data has been scanned from the printed page. The PAPERBYTE bar code representations provide a standardized means of obtaining the data, but they cannot be compared to the convenience of local mass storage devices such as floppy disks, digital cassettes or audio cassettes. Thus if repeated use of the software obtained from bar code is anticipated, we recommend that the user make a copy on some form of magnetic medium.

Bar Code Loader by Ken Budnik, the first in the PAPERBYTE series of software books, provides a brief history of bar codes, a look at the PAPERBYTE bar code format including flowcharts, a general bar code loader algorithm and well documented programs with complete implementation and checkout procedures for 6800, 6502 and 8080/Z-80 based systems.

MONDEB,

AN ADVANCED M6800 MONITOR-DEBUGGER incorporates all the general features of Motorola's MIKBUG monitor as well as numerous other capabilities. While extremely versatile, ease of use was a prime design consideration. The other primary goal was minimum memory requirements while retaining maximum versatility. The size of the entire MONDEB program is less than 3 K.

Some of the command capabilities of MONDEB include displaying and setting the contents of registers, setting interrupts for debugging, testing a programmable memory range for bad memory locations, changing the display and input base of numbers, displaying the contents of memory, searching for a specified string, copying a range of bytes from one location in memory to another, and defining the location to which control will transfer upon receipt of an interrupt.

