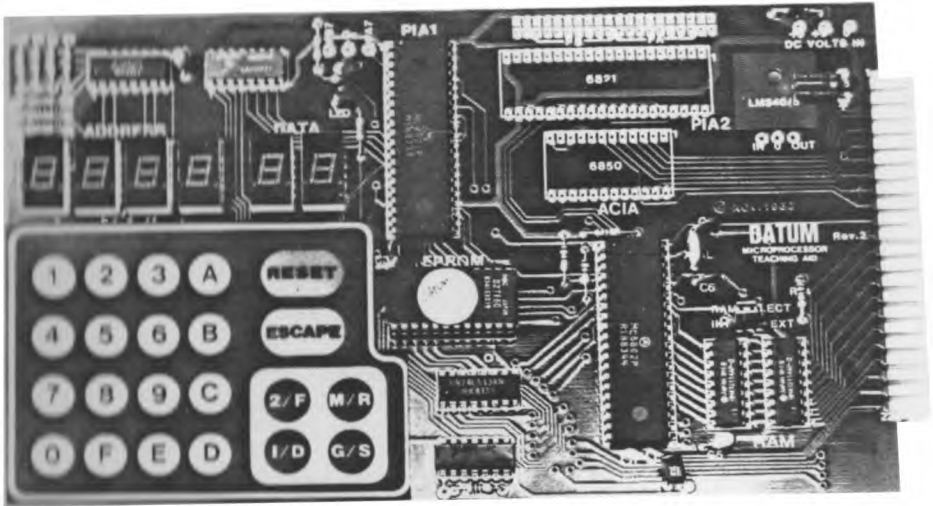


Working with **DATUM**



M. R. Haskard
EDITOR

BOOK 3

NATIONAL LIBRARY OF AUSTRALIA CARD NUMBER AND ISBN : 0 909386 34 X

Copyright © 1988 by Techsearch Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or stored in any form or by any means, electrical, mechanical, photocopying, recording or otherwise without the prior written permission of the publisher.



TECHSEARCH INC.

THE SOUTH AUSTRALIAN INSTITUTE OF TECHNOLOGY
North Torrens Adelaide 5000 phone 723 3866 ext 268

PREFACE

This Book 3 brings together work that has occurred over several years. The DATERM monitor program allows DATUM to be used with a terminal and has been available in house for some time. It is now being made public.

The interpreter BASIC package was developed for two reasons. Firstly, many DATUM applications have been in the control and instrumentation area and a high level language simplifies programming. The terminal is only needed to set up the control system and can be removed once the system is in operation. The second reason for the ROM based BASIC is to enhance the teaching role of DATUM by allowing students to come to grips with a high level language. For example, how is such a language implemented on a simple microprocessor.

The Book has been written in two separate yet related sections, the first covering DATERM and the second BASIC. In keeping with our educational policy complete listings of the monitor and BASIC programs are included.

I trust you will have many hours of experimentation as you learn more about microprocessors and in particular these two ROM based software packages.

Malcolm R. Haskard.
Editor
5th September, 1988.

ACKNOWLEDGEMENTS

I wish to acknowledge the assistance of my colleagues in the writing of these software packages, documentation and testing of them. For DATERM my thanks go to Peter O'Neill who wrote the original software version and to Ian May author of the associated documentation.

In the case of the BASIC interpreter my thanks to Steve Dunn, who commenced the work, Scott Gogler who carried it through to completion, including the documentation and Malcolm Macdonald who was advisor and consultant.

Finally my thanks to John Duval who checked out the software packages and confirmed that they functioned correctly.

Malcolm Haskard

PURCHASING DATUM AND ASSOCIATED SOFTWARE

Should you wish to purchase a DATUM, either of these ROM based software packages or associated documentation, then they can be obtained from:-

GAMMATRON PTY LTD
11 Acrylon Road
Salisbury South 5109
SOUTH AUSTRALIA
Phone (08) 281 2688

SECTION 1

THE DATERM MONITOR PROGRAM

by IAN C. MAY

CONTENTS

	Page
1. Introduction	1
2. Using DATERM	1
3. Monitor Commands	1
3.1 Register commands	2
3.2 Memory commands	3
3.3 Execute commands	5
3.4 Breakpoint commands	5
3.5 Save and load commands	6
3.6 The Trace command	6
4. Monitor Error Messages	8
5. Appendix - Monitor Listing	10

THE DATERM MONITOR PROGRAM

1. Introduction

The DATERM monitor program allows a DATUM system consisting of the main processor card and an extension card (fitted with at least the RS232 interface) to communicate with an RS232 terminal at a baud rate of 300 bits per second. The program incorporates all the features of the original DATUM monitor and more. DATERM was written by P.D. O'NEILL the author of the original DATUM monitor program DATUMON. An assembly language listing of DATERM is provided in the Appendix.

2. Using DATERM

Assuming the terminal and power supply are correctly connected pressing the DATUM's reset key should produce the following sign-on message and the monitor prompt (>). (Of course the version number may be different).

```
-- DATERM --
```

```
V1.3
```

```
>
```

The monitor program is now ready to accept the operator's commands.

3. Monitor commands

DATERM has 18 commands, some of which require the use of the keyboard "control" key. In the text a "^" character is used to signify that the control key must be pressed as well as the letter key following i.e. ^A means press the control and the "A" keys together. If a "^" character appears in the text without another letter immediately following it refers to the up arrow key only. The commands that require use of the control key are :

```
^A - display or change accumulator A
^B - display or change accumulator B
^C - display or change the condition code register
^P - display or change the program counter
^X - display or change the index register
^S - display or change the stack pointer
```

and the commands that do not require use of the control key are:

```
B - set a breakpoint
D - display the contents of memory
F - fill memory with a particular byte
G - go and execute a program
I - inspect memory byte at a time
J - jump to address and execute program
```

- L - load data from cassette
- M - move a block of memory
- R - display all registers
- T - trace user's program
- W - write data to cassette
- X - delete breakpoint

The following paragraphs describe the use of these commands which have been grouped by function. Note that hexadecimal numbers are indicated by the prefix "\$".

3.1 Register commands

The DATERM monitor maintains a set of pseudo registers that represent the internal registers of the 6802. Whenever a user program is executed (using either the G or J commands) the pseudo register values are loaded into the 6802 registers using the return from interrupt instruction (RTI). Hence the operator can begin program execution with known values in the 6802's registers.

To display the contents of the pseudo accumulator A register (ACCA) the key sequence ^A is pressed. The monitor will respond ACC A = XX where XX is the current contents of ACCA. Similarly ^B displays ACCB, ^P the program counter (PC), ^X the index register (IX), and ^S the stack pointer (SP). Note that for the sixteen bit registers (PC,X,& SP) the value is displayed in sixteen bits i.e. PC = XXXX. The command ^C displays the condition code register (CCR) though not as a byte XX but in the 6802 flag format "HINZVC". For instance in response to ^C the monitor program may display CCR = H--Z-C indicating that the H,Z and C flags are set whilst the I,N and V flags are not.

If viewing the contents of a register is all that is required the command can be terminated by pressing the return key. If a change in the contents is desired the appropriate hexadecimal value may be entered. For instance to change ACCA to \$88 the following sequence would be used :

keyboard input	output on screen
^A	>^A ACCA = XX

the contents of ACCA have been displayed, then by entering

88	>^A ACCA = XX 88
----	------------------

ACCA is changed to 88. The monitor then prompts for another command with ">". Note that the correct size of operand must be entered i.e. to set the PC to \$12, the operator must enter 0012. For the condition code register the flag values must be entered. If for example it was desired that the I bit be set then following ^C the data "-I---" should be entered. Entering a "-" clears the flag in the corresponding bit position.

The final register command is "R". This command displays all the 6802 registers but does not allow them to be changed.

Here are some examples of DATERM's register commands.

VDU output	command function
>^A ACC A = 00	display ACCA
>^B ACC B = 00 55	set ACCB to \$55
>^C CCR = -----	display CCR
>^C CCR = ----- HINZVC	set CCR to HINZVC
>^C CCR = HINZVC	display CCR
>^C CCR = HINZVC ----V-	set CCR V flag
>^P PC = 0000	display PC
>^S SP = 0060	display SP
>^X IX = 0000 1234	set IX to 1234
>R ACC A = 00 ACC B = 55 CCR = -----C IX = 0000 SP = 0060 PC = 0000	display registers

3.2 Memory commands

There are 4 commands that operate on the DATUM's memory, D,F,I and M. The D (DUMP) command displays a block of memory on the screen in both hexadecimal and ASCII formats. The operands of this command are a start and an end address. The start address is rounded down to the nearest XXX0 and the end address rounded up to the nearest XXXF to produce a more readable display. For example to display the contents of the memory between \$1000 and \$107F the following could be entered:

>D 1000 107F

or alternatively >D 100F 1070 due to the effect of the rounding.
The output from such a command is :

>D 1000 107F

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
1000	3F	D3	02	FF	00	FF	00	7F	88	FF	80	7F	10	DF	00	FF	?.....~....~....
1010	82	FF	00	6B	04	FB	00	FF	00	FF	01	FB	24	7F	40	FF	...k.....\$~@.
1020	00	FF	00	7F	00	FB	80	FF	04	EF	00	EF	40	EB	00	FF	...~.....@...
1030	40	FD	08	D7	10	FF	00	FF	84	EF	00	FF	01	FF	10	FF	@.....~....~....
1040	00	DF	00	BF	00	DF	00	7F	04	FF	02	FF	00	F7	00	7F~.....~....
1050	00	FB	20	FF	02	FF	00	FF	00	FF	02	EF	14	BE	00	FF~.....~....
1060	20	FF	00	FF	02	7F	04	F7	02	DF	10	FF	40	BF	30	FF~.....@.0.
1070	00	FF	84	AF	00	FF	04	FF	00	FF	40	FF	00	7F	00	F5~.....@...~..

>

If a memory byte does not correspond to a printable ASCII character it is shown as a "." in the ASCII section of the memory dump. If the memory dump covers more than one page of memory the monitor will pause following the first page until the escape key

is pressed (which will continue the dump) or the return key pressed (which terminates the dump).

The FILL (F) command fills the memory between two given memory addresses with a given data byte. This command can be used to fill a data area with zeros for example or a text area with ASCII blanks. Below is an example of the fill command.

```
>F 1000 101F 55    Fill memory between $1000 and $101F with $55
>D 1000 101F      and display the contents with Dump.
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
1000	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	UUUUUUUUUUUUUUUUUU
1010	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	UUUUUUUUUUUUUUUUUU

>

The Inspect memory command (I) allows the memory to be examined and optionally altered on a byte by byte basis. The I command requires as operand the first memory address to be examined, e.g. to examine the data at memory address \$1200 the command line >I 1200 would be entered. The monitor will respond with the address and data byte at that address. To move to the next address without changing the data press the space bar. To enter a new hexadecimal value at the current address press the desired hexadecimal keys. If you wish to enter an ASCII value precede the character with a "'" i.e. 'A enters an ASCII capital A. To move backwards a memory location press the "^" key and to terminate the command press the return key.

Example of I command usage:

```
>I 1200          Inspect the memory at $1200
1200 58          contents of $1200 is $58/ press space
1201 69          contents of $1201 is $69/ press space
1202 00 ^        $1202 contains 00/ move back a byte
1201 69 ^        contents of $1201/ move back a byte
1200 58          contents of $1200/ press space
1201 69          contents of $1201/ press space
1202 00          contents of $1202/ press space
1203 F7 22       contents of $1203/ change to $22
1204 10 99       contents of $1204/ change to $99
1205 FF ^        contents of $1205/ move back a byte
1204 99 ^        check that $1204 is $99/ move back a byte
1203 22          check that $1203 is $22/ press space
1204 99          contents of $1204/ press space
1205 FF 'A       contents of $1205/ change to ASCII "A"
1206 64 ^        contents of $1206/ move back a byte
1205 41          check $1205 is an A ($41)/ type return
>               command terminated/ new prompt issued
```

The final memory command is Move (M). This command moves a block of memory to another memory area. The operands required are the source start and end addresses and the destination start address. The source and destination blocks may overlay each other without the data being corrupted. Here is an example.

```
>M 1000 101F 1030    move the memory from $1000 to $101F to $1030.
>D 1000 104F          display the contents of memory.
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
1000	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	UUUUUUUUUUUUUUUUUUUU
1010	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	UUUUUUUUUUUUUUUUUUUU
1020	00	FF	00	7F	00	FB	80	FF	04	EF	00	EF	40	EB	00	FF	...~.....@...
1030	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	UUUUUUUUUUUUUUUUUUUU
1040	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	UUUUUUUUUUUUUUUUUUUU

>

As can be seen from the memory dump, locations \$1000 through \$101F have been copied to locations \$1030 through \$104F.

3.3 Execute commands

There are two commands which can start the execution of a user program, the Go (G) and Jump (J) commands. The only difference between the two commands is that the J command has an operand of the program start address while the G command uses the current pseudo program counter address as start address. Both commands otherwise operate identically by loading the 6802's registers with the pseudo register values and then commencing the execution of the user program. If the user's program started at memory location \$1000, using the G command the operator should first set the pseudo PC to \$1000 using the ^P command and then issue the G command. To use the J command J 1000 would be entered. The J command is typically used to initially start execution of a program, while the G command is used to continue execution following a breakpoint.

3.4 Breakpoint commands

There are two monitor commands concerned with breakpoints, the place breakpoint command B and the clear breakpoint command X. The B command has one operand which is the address of the breakpoint. The X command has no operand. The place breakpoint command works by first reading the program byte at the address of the operand which is stored in the monitor scratchpad. The program byte is then replaced by a software interrupt instruction (SWI - \$3F) and then the address is stored in the scratchpad. When the SWI instruction is executed (actually any SWI instruction) program control is returned to the monitor which then prints the register contents on the screen. The clear breakpoint command replaces the program byte stored in the monitor scratchpad area thus removing the breakpoint.

Here is an example of the breakpoint commands.

```
>B 1000                set a breakpoint at $1000
>^P PC = 0000 1000    set the pseudo PC to $1000
>G ACC A = 00  ACC B = 55  CCR = ----C start program with G
  IX = 0000    SP = 0060   PC = 1000  register dump from SWI
>X                    clear the breakpoint at $1000
```

3.5 Save and Load commands

The cassette save and load commands operate in exactly the same way as those of the DATUMON monitor program and so cassette files are interchangeable between monitor programs. To load a file from cassette simply enter the Load command (L) and then start the cassette player. To save a program enter the Write command (W) followed by the start and end addresses of the memory block you wish to save. The cassette should be put into record mode just before entering the final digit of the end address to allow time for the tape to get up to speed. When writing to tape the cassette data is echoed onto the terminal screen because the ACIA chip is shared between the monitor and the cassette interface.

Examples

```
>L                    load a file from cassette
>W 1000 101F          save memory locations 1000-101F
```

3.6 The Trace command

The final command of the DATERM monitor is Trace (T). This is basically the same as the trace program of the DATUMON monitor but has been customised to suit a VDU. The trace command has a single operand - the program start address. After entering the start address the monitor lists on the screen the hex codes for the instruction it is about to trace. If the space bar is pressed the instruction is executed and a register dump is given along with the next instruction. The register manipulation commands ^A, ^B, ^C, ^P OR ^X may be used while in trace mode to set any of the registers. Note that you cannot set the stack pointer or execute any of the other commands while in trace mode. To exit trace mode press the return key. A typical output is shown below when the following program is traced.

```
1000 86 01          LDAA £1          Load ACCA with 1
1002 C6 09          LDAB £9          Load ACCB with 9
1004 1B             ABA              add ACCB to ACCA
1005 19             DAA              convert to BCD
1006 3F             SWI

>T 1000            Enter trace mode and begin at $1000

1000 86 01          next instruction is LDAA 01/ press space
  ACC A = 01  ACC B = 00  CCR = -I----
  IX = 1000   SP = 0060   PC = 1002
```

```

1002 C6 09          next instruction is LDAB 09/ press space
  ACC A = 01  ACC B = 09  CCR = -I----
  IX = 1000   SP = 0060   PC = 1004
1004 1B           next instruction is ABA / press space
  ACC A = 0A  ACC B = 09  CCR = -I----
  IX = 1000   SP = 0060   PC = 1005
1005 19           next instruction is DAA / press space
  ACC A = 10  ACC B = 09  CCR = -I----
  IX = 1000   SP = 0060   PC = 1006
1006 3F           next instruction is SWI / press space

>                  exit trace mode

```

Note that an SWI instruction cannot be traced nor can a program in ROM.

4. Monitor error messages

If an illegal command is entered the response "WHAT?" is printed on the terminal and a new prompt will be given. This message will also be issued if an illegal entry is made for a numeric operand. In memory inspect mode using the I command, if data is written to memory it is immediately checked. If the correct data is not read back the message "NOT R/W MEMORY!" will be issued. If a bad checksum is encountered during a cassette load command (L) the message "LOAD ERROR" will be issued.

>D 1000 107F

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
1000	3F	D3	02	FF	00	FF	00	7F	88	FF	80	7F	10	DF	00	FF	?.....~.....
1010	82	FF	00	6B	04	FB	00	FF	00	FF	01	FB	24	7F	40	FF	...k.....~..
1020	00	FF	00	7F	00	FB	80	FF	04	EF	00	EF	40	EB	00	FF	...~.....~..
1030	40	FD	08	D7	10	FF	00	FF	84	EF	00	FF	01	FF	10	FF~.....
1040	00	DF	00	BF	00	DF	00	7F	04	FF	02	FF	00	F7	00	7F~.....
1050	00	FB	20	FF	02	FF	00	FF	00	FF	02	EF	14	BE	00	FF~.....
1060	20	FF	00	FF	02	7F	04	F7	02	DF	10	FF	40	BF	30	FF~.....
1070	00	FF	84	AF	00	FF	04	FF	00	FF	40	FF	00	7F	00	F5~.....

>

>F 1000 101F 55 Fill memory between \$1000 and \$101F with \$55
 >D 1000 101F and display the contents with Dump.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
1000	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	UUUUUUUUUUUUUUUU
1010	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	UUUUUUUUUUUUUUUU

>

>M 1000 101F 1030 move the memory from \$1000 to \$101F to \$1030.
 >D 1000 104F display the contents of memory.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
1000	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	UUUUUUUUUUUUUUUU
1010	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	UUUUUUUUUUUUUUUU
1020	00	FF	00	7F	00	FB	80	FF	04	EF	00	EF	40	EB	00	FF	...~.....~..
1030	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	UUUUUUUUUUUUUUUU
1040	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	UUUUUUUUUUUUUUUU

>

5. APPENDIX - MONITOR LISTING

* DATERM IS THE MONITOR PROGRAM
* FOR A SERIAL V.D.U.

* SYSTEM EQUATES

0004	EOT	EQU	4	
000A	LF	EQU	\$A	
000D	CR	EQU	\$D	
001B	ESC	EQU	\$1B	
003E	PROMPT	EQU	'>	
0020	FORWRD	EQU	\$20	INCREMENT MEMORY DISP.
005E	REVERS	EQU	'^	DECREMENT MEMORY DISP.
0027	ASCII	EQU	' '	PUT ASCII CHAR. IN MEM.
0040	CNTRL	EQU	\$40	
007F	STACK	EQU	\$7F	SYSTEM STACK
0060	USTACK	EQU	\$60	USER STACK
4000	ACIAC	EQU	\$4000	ACIA CONTROL
4001	ACIAD	EQU	\$4001	ACIA DATA
3F00	DACDAT	EQU	\$3F00	D TO A ADDRESS

* TEMP STORAGE

0000		ORG	0	
0000	NMIV	RMB	2	PSEUDO NMI VECTOR
0002	SWIV	RMB	2	PSEUDO SWI VECTOR
0004	IRQV	RMB	2	PSEUDO IRQ VECTOR
0006	XTEMP	RMB	2	INDEX TEMP STORAGE
0008	XTEMP1	RMB	2	; " " "
000A	XTEMP2	RMB	2	; " " "
000C	XTEMP3	RMB	2	; " " "
000E	STARAD	RMB	2	START ADDRESS
0010	ENDADD	RMB	2	END ADDRESS
0012	TEMP1	RMB	1	ACC. TEMP. STORAGE
0013	CHKSUM	RMB	1	TAPE CHECK SUM
0014	BYTNUM	RMB	1	TAPE BYTE COUNT
0015	CCR	RMB	1	STORAGE FOR USER'S
0016	ACCB	RMB	1	VALUES IN REGISTERS
0017	ACCA	RMB	1	
0018	XH	RMB	2	
001A	PCH	RMB	2	
001C	SPH	RMB	2	
001E	SPH2	RMB	2	
0020	BRPTST	RMB	6	BREAK-POINT STORAGE
0026	TRFLAG	RMB	1	TRACE FLAG
0027	BRAFLG	RMB	1	BRANCH FLAG
0028	INSTR	RMB	3	INSTRUCTION STORAGE
002B	OPCODE	RMB	1	OPCODE STORAGE
002C	BPFLAG	RMB	1	BREAK POINT FLAG
002D	LDFLG	RMB	1	CASSETTE LOAD FLAG
7000		ORG	\$7000	

* SWI SERVICE				
7000 DE 02	SWI	LDX	SWIV	PSEUDO VECTOR FOR SWI
7002 6E 00		JMP	0,X	
* NMI SERVICE				
7004 DE 00	NMI	LDX	NMIV	PSEUDO VECTOR FOR NMI
7006 6E 00		JMP	0,X	
* RESET SERVICE				
7008 7E 70 F4	RESET	JMP	START1	JUMP TO RESTART
* IRQ SERVICE				
700B DE 04	IRQ	LDX	IRQV	PSEUDO VECTOR FOR IRQ
700D 6E 00		JMP	0,X	
* GET ADDRESS				
700F 8D 09	GETADD	BSR	GETBYT	GET HIGH ORDER BYTE
7011 97 08		STA A	XTEMP1	SAVE IT
7013 8D 05		BSR	GETBYT	GET LOW ORDER BYTE
7015 97 09		STA A	XTEMP1+1	SAVE IT
7017 DE 08		LDX	XTEMP1	PUT IT IN X REG.
7019 39		RTS		
* INPUT A BYTE				
701A 8D 7B	GETBYT	BSR	GETHEX	GET TWO ASCII CHARACTERS
701C 48	GETBY2	ASL A		AND FORM A BYTE
701D 48		ASL A		RETURN TO CONTROL IF
701E 48		ASL A		AN ERROR IS DETECTED
701F 48		ASL A		
7020 16		TAB		
7021 8D 74		BSR	GETHEX	
7023 1B		ABA		
7024 16		TAB		
7025 DB 13		ADD B	CHKSUM	UPDATE CHECKSUM FOR TAPE
7027 D7 13		STA B	CHKSUM	
7029 39		RTS		
* OUTHL				
702A 44	OUTHL	LSR A		OUTPUT THE LEFTHAND
702B 44		LSR A		NIBBLE OF A BYTE
702C 44		LSR A		
702D 44		LSR A		
702E 84 0F	OUTHR	AND A	£\$F	OUTPUT THE RIGHTHAND
7030 8B 30		ADD A	£\$30	NIBBLE OF A BYTE
7032 81 39		CMP A	£\$39	
7034 23 02		BLS	OUTHR1	
7036 8B 07		ADD A	£\$07	
7038 7E 70 E7	OUTHR1	JMP	OUT	

* PRINT STRING POINTED TO BY INDEX REG.

703B	BD	70	E7	PSTRN0	JSR	OUT	
703E	08				INX		
703F	A6	00		PSTRNG	LDA A	0,X	
7041	81	04			CMP A	£EOT	
7043	26	F6			BNE	PSTRN0	
7045	39				RTS		

* CHANGE MEMORY

7046	BD	70	0F	INSPEC	JSR	GETADD	GET POINTER ADDRESS
7049	DF	0C			STX	XTEMP3	
704B	CE	76	B6	INSP5	LDX	£MMSG3	
704E	8D	EF			BSR	PSTRNG	
7050	CE	00	0C		LDX	£XTEMP3	
7053	8D	6C			BSR	OUT4HS	OUTPUT THE ADDRESS
7055	DE	0C			LDX	XTEMP3	
7057	8D	6A			BSR	OUT2HS	OUTPUT CONTENTS AT THAT ADDR
7059	DF	0C			STX	XTEMP3	
705B	BD	70	DA		JSR	IN	
705E	81	20			CMP A	£FORWRD	INCREMENT ADDRESS POINTER?
7060	27	E9			BEQ	INSP5	
7062	81	5E			CMP A	£REVERS	DECREMENT ADDRESS POINTER?
7064	27	0F			BEQ	INSP7	
7066	81	27			CMP A	£ASCII	INPUT ASCII ?
7068	27	28			BEQ	INSP9	
706A	81	0D			CMP A	£CR	
706C	27	21			BEQ	INSP8	
706E	8D	33			BSR	GETH2	
7070	8D	AA			BSR	GETBY2	
7072	09				DEX		
7073	20	06			BRA	INSP6	
7075	09			INSP7	DEX		
7076	09				DEX		
7077	DF	0C			STX	XTEMP3	
7079	20	D0			BRA	INSP5	
707B	A7	00		INSP6	STA A	0,X	
707D	A1	00			CMP A	0,X	
707F	27	CA			BEQ	INSP5	
7081	BD	74	02		JSR	PCRLF	
7084	CE	76	BA		LDX	£MMSG4	MEMORY READ ERROR
7087	BD	70	3F		JSR	PSTRNG	
708A	20	03			BRA	INSP8	
708C	7E	71	5B	WHAT1	JMP	WHAT	PRINT 'WHAT?'
708F	7E	71	1B	INSP8	JMP	CONTRL	RETURN TO CONTROL
7092	8D	46		INSP9	BSR	IN	
7094	09				DEX		
7095	20	E4			BRA	INSP6	

* INPUT HEX CHARACTER

7097	BD	70	DA	GETHEX	JSR	IN	INPUT A CHARACTER
709A	7D	00	2D		TST	LDFLG	IS LOAD FLAG SET

709D 26 04		BNE	GETH2	
709F 81 0D		CMP A	£CR	
70A1 27 EC		BEQ	INSP8	
70A3 80 30	GETH2	SUB A	£\$30	STRIP OFF 30 HEX
70A5 2B E5		BMI	WHAT1	TEST IF IT IS CNTRL CHR.
70A7 81 09		CMP A	£9	NO THEN TEST IF > 9
70A9 2F 0A		BLE	GETH3	IF SO RETURN
70AB 81 11		CMP A	£\$11	IF NOT CHECK FOR A - F
70AD 2B DD		BMI	WHAT1	IF < A THEN PRINT 'WHAT?'
70AF 81 16		CMP A	£\$16	IS IT > F
70B1 2E D9		BGT	WHAT1	YES THEN PRINT 'WHAT?'
70B3 80 07		SUB A	£7	ADJUST FOR A - F
70B5 39	GETH3	RTS		

* OUTPUT 2 HEX CHARACTERS

70B6 A6 00	OUT2H	LDA A	0,X
70B8 BD 70 2A		JSR	OUTH1
70BB A6 00		LDA A	0,X
70BD 08		INX	
70BE 7E 70 2E		JMP	OUTHR

* OUTPUT 4 HEX CHARACTERS AND A SPACE

70C1 8D F3	OUT4HS	BSR	OUT2H
------------	--------	-----	-------

* OUTPUT 2 HEX CHARACTERS AND A SPACE

70C3 8D F1	OUT2HS	BSR	OUT2H
------------	--------	-----	-------

* OUTPUT A SPACE

70C5 86 20	OUTS	LDA A	£\$20
70C7 7E 70 E7		JMP	OUT

* PAUSE

70CA 8D 0E	PAUSE	BSR	IN	TEST FOR CR OR ESC
70CC 81 1B		CMP A	£ESC	AND REPORT THE STATUS
70CE 26 03		BNE	PAUSE2	CR => CARRY=1 ACCA=0
70D0 4F		CLR A		ESC => CARRY=0 ACCA=0
70D1 0C		CLC		NEITHER => CARRY=1 AND
70D2 39		RTS		ACCA NOT = 0
70D3 81 0D	PAUSE2	CMP A	£CR	
70D5 26 01		BNE	PAUSE3	
70D7 4F		CLR A		
70D8 0D	PAUSE3	SEC		
70D9 39		RTS		

* INPUT ONE CHARACTER

70DA 37	IN	PSH B		SAVE ACC B
70DB B6 40 00	IN1	LDA A	ACIAC	TEST IF BUFFER FULL
70DE 47		ASR A		IS IT FULL?
70DF 24 FA		BCC	IN1	WAIT TILL IT IS

70E1	B6	40	01	LDA	A	ACIAD	READ IN THE BYTE
70E4	84	7F		AND	A	£\$7F	MASK PARITY
70E6	33			PUL	B		

* OUTPUT ONE CHARACTER

70E7	37			OUT	PSH	B	
70E8	F6	40	00	OUT1	LDA	B	ACIAC
70EB	57				ASR	B	
70EC	57				ASR	B	
70ED	24	F9			BCC		OUT1
70EF	B7	40	01		STA	A	ACIAD
70F2	33				PUL	B	
70F3	39				RTS		

* POWER ON SEQUENCE

70F4	8E	00	7F	START1	LDS	£STACK	
70F7	86	03			LDA	A	£3
70F9	B7	40	00		STA	A	ACIAC
70FC	86	15			LDA	A	£\$15
70FE	B7	40	00		STA	A	ACIAC
7101	CE	71	1B		LDX	£CONTRL	
7104	DF	00			STX	NMIV	
7106	CE	00	15		LDX	£CCR	POINT TO CCR AND CLEAR
7109	4F				CLR	A	
710A	A7	00		INIT	STA	A	0,X INITIALIZE USER REGISTER STORE
710C	08				INX		
710D	8C	00	20		CPX	£CCR+11	
7110	26	F8			BNE	INIT	
7112	BD	73	EC		JSR	DELBP	INITIALIZE BREAK-POINT STORE
7115	CE	77	0F		LDX	£MSG13	OUTPUT SIGN ON MESSAGE
7118	BD	70	3F		JSR	PSTRNG	
711B	8E	00	7F	CONTRL	LDS	£STACK	SET THE STACK
711E	7F	00	2D		CLR	LDFLG	CLEAR THE LOAD FLAG
7121	86	15			LDA	A	£\$15
7123	B7	40	00		STA	A	ACIAC
7126	7F	00	26		CLR	TRFLAG	RESET THE TRACE FLAG
7129	DE	1E			LDX	SPH2	CHECK USER STACK
712B	8C	00	00		CPX	£0	
712E	26	05			BNE	NOINIT	IF NON-ZERO DON'T CHANGE IT
7130	CE	00	60		LDX	£USTACK	IF ZERO SET USTACK
7133	DF	1E			STX	SPH2	
7135	CE	76	AF	NOINIT	LDX	£MSG1	OUTPUT A CR,LF, AND A PROMPT
7138	BD	70	3F		JSR	PSTRNG	
713B	CE	75	3F		LDX	£RETSWI	SET SWI TO DISPLAY REGISTERS
713E	DF	02			STX	SWIV	
7140	BD	70	DA		JSR	IN	WAIT FOR A COMMAND
7143	81	1F			CMP	A	£\$1F IS IT A CONTROL CHARACTER
7145	23	05			BLS	CTLCHR	IF SO OUTPUT A SPACE TO
7147	36				PSH	A	KEEP THE DISPLAY TIDY
7148	BD	70	C5		JSR	OUTS	
714B	32				PUL	A	
714C	CE	77	4F	CTLCHR	LDX	£FUTABL	POINT TO FUNCTION TABLE
714F	A1	00		NXTCHR	CMP	A	0,X IS IT A VALID COMMAND

7151	27	10		BEQ	MATCH	YES GOTO MATCH
7153	08			INX		NO TRY NEXT ONE
7154	08			INX		
7155	08			INX		
7156	8C	77	88	CPX	£TABLEN	IS IT THE END OF THE TABLE
7159	26	F4		BNE	NXTCHR	NO TRY ANOTHER
715B	CE	76	CA	LDX	£MESG5	PRINT 'WHAT?'
715E	BD	70	3F	JSR	PSTRNG	
7161	20	B8		BRA	CONTRL	GO AND WAIT FOR ANOTHER
7163	EE	01	MATCH	LDX	1,X	POINT TO FUNCTION SUBROUTINE
7165	AD	00		JSR	0,X	JUMP TO IT
7167	20	B2		BRA	CONTRL	

* DUMP THE CONTENTS OF MEMORY

7169	BD	72	3A	DUMP	JSR	LIMITS	GET THE START AND END ADDR.
716C	96	0F			LDA A	STARAD+1	CLEAN UP START AND END
716E	84	F0			AND A	£\$F0	ADDRESSES SO THAT THE
7170	97	0F			STA A	STARAD+1	DISPLAY WILL BE NEAT
7172	48				ASL A		
7173	48				ASL A		
7174	48				ASL A		
7175	48				ASL A		
7176	97	12			STA A	TEMP1	
7178	96	11			LDA A	ENDADD+1	
717A	8A	0F			ORA A	£\$0F	
717C	97	11			STA A	ENDADD+1	
717E	DE	10			LDX	ENDADD	
7180	08				INX		
7181	DF	10			STX	ENDADD	
7183	20	03			BRA	NEXPA1	
7185	7F	00	12	NEXPAG	CLR	TEMP1	
7188	BD	74	02	NEXPA1	JSR	PCRLF	OUTPUT THE HEADING FOR
718B	BD	74	02		JSR	PCRLF	A NEW PAGE
718E	CE	76	69		LDX	£MESG0	
7191	BD	70	3F		JSR	PSTRNG	
7194	BD	74	02		JSR	PCRLF	
7197	5F			NEXLIN	CLR B		OUTPUT THE BASE ADDRESS
7198	CE	00	0E		LDX	£STARAD	FOR THE NEXT 16 BYTES
719B	BD	70	C1		JSR	OUT4HS	
719E	BD	70	C5		JSR	OUTS	
71A1	DE	0E			LDX	STARAD	
71A3	BD	70	C3	NEXBYT	JSR	OUT2HS	OUTPUT 16 HEX BYTES
71A6	5C				INC B		WITH A SPACE BETWEEN EACH
71A7	17				TBA		AND TWO SPACES BETWEEN THE
71A8	84	03			AND A	£3	BASE ADDR. AND THE FIRST
71AA	26	03			BNE	NOT2	BYTE, 3RD AND 4TH BYTES,
71AC	BD	70	C5		JSR	OUTS	7TH AND 8TH BYTES, BTH AND
71AF	C1	10		NOT2	CMP B	£\$10	CTH BYTES AND FTH BYTE AND
71B1	26	F0			BNE	NEXBYT	THE ASCII
71B3	DE	0E			LDX	STARAD	
71B5	5F				CLR B		
71B6	A6	00		ASCII	LDA A	0,X	THE ASCII EQUIVALENT TO
71B8	81	7F			CMP A	£\$7F	THE HEX IS DISPLAYED
71BA	22	09			BHI	DOT	IF THE BYTE IS=>\$20 AND

71BC	81	1F		CMP A	£\$1F	<=\$7F THE THAT CHR.
71BE	23	05		BLS	DOT	WILL BE PRINTED, IF NOT
71C0	BD	70	E7	JSR	OUT	THEN A '.' WILL BE PRINTED
71C3	20	05		BRA	CHKEND	
71C5	86	2E		LDA A	£\$2E	
71C7	BD	70	E7	JSR	OUT	
71CA	08			CHKEND	INX	
71CB	5C				INC B	
71CC	C1	10		CMP B	£\$10	
71CE	26	E6		BNE	ASCII	LAST ASCII?
71D0	DF	0E		STX	STARAD	
71D2	BD	74	02	JSR	PCRLF	
71D5	D6	12		LDA B	TEMP1	
71D7	5C				INC B	
71D8	D7	12		STA B	TEMP1	
71DA	DE	0E		LDX	STARAD	
71DC	9C	10		CPX	ENDADD	
71DE	26	01		BNE	TEST10	
71E0	39			RTS		
71E1	C1	10		TEST10	CMP B	£\$10
71E3	26	B2		BNE	NEXLIN	LAST LINE IN PAGE?
71E5	BD	70	CA	HANGUP	JSR	PAUSE
71E8	26	FB		BNE	HANGUP	WAIT FOR 'ESC' BEFORE
71EA	25	03		BCS	EXIT	GOING TO NEXT PAGE
71EC	7E	71	85		JMP	NEXPAG
71EF	7E	71	1B	EXIT	JMP	CONTRL

* JUMP TO AN ADDRESS

71F2	BD	70	0F	JUMP	JSR	GETADD	GET JUMP ADDRESS
71F5	6E	00			JMP	0,X	JUMP TO IT

* UPDAS2, UPDATE USER STACK POINTER

71F7	CE	00	1C	UPDAS2	LDX	£SPH	UPDATE SPH
71FA	A6	03			LDA A	3,X	AND ADJUST THE DISPLAYED
71FC	80	07			SUB A	£7	VALUE SO THAT IT IS
71FE	A7	01			STA A	1,X	CORRECT, I.E. NOT DISPLACED
7200	A6	02			LDA A	2,X	BY SEVEN BYTES DUE TO THE RTI
7202	82	00			SBC A	£0	
7204	A7	00			STA A	0,X	
7206	39				RTS		

* GORTI, GOTO USER PROGRAM WITH DEFINED REGISTERS

7207	8D	EE		GORTI	BSR	UPDAS2	UPDATE SPH
7209	DE	1C			LDX	SPH	PUT ALL THE REGS. ONTO
720B	96	15			LDA A	CCR	THE STACK READY FOR THE
720D	A7	01			STA A	1,X	RTI
720F	96	16			LDA A	ACCB	
7211	A7	02			STA A	2,X	
7213	96	17			LDA A	ACCA	
7215	A7	03			STA A	3,X	
7217	96	18			LDA A	XH	

7219	A7	04	STA	A	4,X
721B	96	19	LDA	A	XH+1
721D	A7	05	STA	A	5,X
721F	96	1A	LDA	A	PCH
7221	A7	06	STA	A	6,X
7223	96	1B	LDA	A	PCH+1
7225	A7	07	STA	A	7,X
7227	9E	1C	LDS		SPH
7229	3B		RTI		

* FILL MEMORY WITH A GIVEN BYTE

722A	8D	0E	FILL	BSR	LIMITS	GET START AND END ADDR.
722C	BD	70 1A		JSR	GETBYT	GET FILL BYTE
722F	DE	0E		LDX	STARAD	PUT THAT BYTE INTO
7231	09			DEX		ALL MEMORY LOCATIONS
7232	08		FILL1	INX		BETWEEN THE START AND
7233	A7	00		STA	A 0,X	THE END.
7235	9C	10		CPX	ENDADD	
7237	26	F9		BNE	FILL1	
7239	39			RTS		

* LIMITS, GET STARTING AND ENDING ADDRESSES

723A	BD	70 0F	LIMITS	JSR	GETADD	GET START ADDRESS
723D	DF	0E		STX	STARAD	SAVE IT
723F	BD	70 C5		JSR	OUTS	OUTPUT A SPACE
7242	BD	70 0F		JSR	GETADD	GET END ADDRESS
7245	DF	10		STX	ENDADD	SAVE IT
7247	BD	70 C5		JSR	OUTS	OUTPUT A SPACE
724A	39			RTS		

* MOVE A BLOCK OF MEMORY

724B	8D	ED	MOVE	BSR	LIMITS	GET START AND END ADDR.
724D	BD	70 0F		JSR	GETADD	GET DESTINATION ADDR.
7250	DF	06		STX	XTEMP	SAVE IT
7252	96	0E		LDA	A STARAD	NOW DETERMINE THE
7254	90	06		SUB	A XTEMP	DIRECTION OF THE BLOCK
7256	22	2F		BHI	BLOCHI	MOVE.
7258	96	0F		LDA	A STARAD+1	
725A	90	07		SUB	A XTEMP+1	
725C	22	29		BHI	BLOCHI	
725E	96	11		LDA	A ENDADD+1	
7260	D6	10		LDA	B ENDADD	
7262	90	0F		SUB	A STARAD+1	
7264	D2	0E		SBC	B STARAD	
7266	9B	07		ADD	A XTEMP+1	
7268	D9	06		ADC	B XTEMP	
726A	97	07		STA	A XTEMP+1	
726C	D7	06		STA	B XTEMP	
726E	DE	0E		LDX	STARAD	
7270	09			DEX		
7271	DF	0E		STX	STARAD	
7273	DE	10	DECLP	LDX	ENDADD	PERFORM A DECREMENTING

7275	A6	00		LDA A	0,X	BLOCK MOVE.
7277	09			DEX		
7278	DF	10		STX	ENDADD	
727A	9C	0E		CPX	STARAD	
727C	27	22		BEQ	LAST	IS IT THE LAST BYTE?
727E	DE	06		LDX	XTEMP	
7280	A7	00		STA A	0,X	
7282	09			DEX		
7283	DF	06		STX	XTEMP	
7285	20	EC		BRA	DECLP	
7287	DE	10	BLOCHI	LDX	ENDADD	
7289	08			INX		
728A	DF	10		STX	ENDADD	
728C	DE	0E	INCLP	LDX	STARAD	PERFORM AN INCREMENTING
728E	A6	00		LDA A	0,X	BLOCK MOVE.
7290	08			INX		
7291	DF	0E		STX	STARAD	
7293	9C	10		CPX	ENDADD	
7295	27	09		BEQ	LAST	IS IT THE LAST BYTE?
7297	DE	06		LDX	XTEMP	
7299	A7	00		STA A	0,X	
729B	08			INX		
729C	DF	06		STX	XTEMP	
729E	20	EC		BRA	INCLP	
72A0	DE	06	LAST	LDX	XTEMP	
72A2	A7	00		STA A	0,X	
72A4	39			RTS		

* DISPLAY/CHANGE ACCA

72A5	CE	76	D2	ADPCN	LDX	%MSG6A	PRINT '^A = '
72A8	BD	70	3F		JSR	PSTRNG	
72AB	CE	00	17		LDX	%ACCA	POINT TO ACCA STORE
72AE	-20	3C			BRA	DPCN	

* DISPLAY/CHANGE ACCB

72B0	CE	76	DE	BDPCN	LDX	%MSG7A	PRINT '^B = '
72B3	BD	70	3F		JSR	PSTRNG	
72B6	CE	00	16		LDX	%ACCB	POINT TO ACCB STORE
72B9	20	31			BRA	DPCN	

* DISPLAY/CHANGE CCR

72BB	CE	77	05	CDPCN	LDX	%MSG1A	PRINT '^C = '
72BE	BD	70	3F		JSR	PSTRNG	
72C1	CE	00	15		LDX	%CCR	POINT TO CCR STORE
72C4	BD	73	0C		JSR	DISCCR	
72C7	BD	73	33		JSR	GETCCR	
72CA	39				RTS		

* DISPLAY/CHANGE PC

72CB	CE	76	F3	PDPCN	LDX	%MSG9A	PRINT '^P = '
72CE	BD	70	3F		JSR	PSTRNG	

72D1	CE	00	1A	LDX	£PCH	POINT TO PC STORE
72D4	20	23		BRA	DPCN1	

* DISPLAY/CHANGE SP

72D6	CE	76	FC	SDPCN	LDX	£MESG0A	PRINT '^S = '
72D9	BD	70	3F		JSR	PSTRNG	
72DC	CE	00	1E		LDX	£SPH2	POINT TO SP STORE
72DF	20	18			BRA	DPCN1	

* DISPLAY/CHANGE X

72E1	CE	76	EA	XDPCN	LDX	£MESG8A	PRINT '^X = '
72E4	BD	70	3F		JSR	PSTRNG	
72E7	CE	00	18		LDX	£XH	POINT TO X STORE
72EA	20	0D			BRA	DPCN1	

* CHANGE A BYTE

72EC	DF	06		DPCN	STX	XTEMP	SAVE X
72EE	BD	70	C3		JSR	OUT2HS	OUTPUT CURRENT BYTE
72F1	BD	70	1A		JSR	GETBYT	GET NEW BYTE
72F4	DE	06			LDX	XTEMP	RESTORE X
72F6	A7	00			STA A	0,X	SAVE NEW VALUE
72F8	39				RTS		

* CHANGE A WORD

72F9	DF	06		DPCN1	STX	XTEMP	SAVE X
72FB	BD	70	C1		JSR	OUT4HS	OUTPUT CURRENT WORD
72FE	BD	70	0F		JSR	GETADD	GET NEW WORD
7301	DE	06			LDX	XTEMP	RESTORE X
7303	96	08			LDA A	XTEMP1	SAVE NEW VALUE
7305	A7	00			STA A	0,X	
7307	96	09			LDA A	XTEMP1+1	
7309	A7	01			STA A	1,X	
730B	39				RTS		

* DISPLAY CCR

730C	A6	00		DISCCR	LDA A	0,X	GET CURRENT CCR
730E	48				ASL A		DELETE BITS 6 AND 7
730F	48				ASL A		
7310	CE	77	88		LDX	£FLAGS	POINT TO FLAG TABLE
7313	48			NXTFLG	ASL A		
7314	25	04			BCS	SETBIT	IS IT SET?
7316	8D	13			BSR	BLANK	IF NOT OUTPUT A BLANK
7318	20	07			BRA	INCPTR	
731A	36			SETBIT	PSH A		OUTPUT FLAG LETTER
731B	A6	00			LDA A	0,X	
731D	BD	70	E7		JSR	OUT	
7320	32				PUL A		
7321	08			INCPTR	INX		POINT TO NEXT FLAG
7322	8C	77	8E		CPX	£FLGEND+1	IS IT THE LAST?
7325	26	EC			BNE	NXTFLG	NO, DO ANOTHER

7327	BD	70	C5	JSR	OUTS	YES, FINISH
732A	39			RTS		

* OUTPUT A BLANK "-"

732B	36			BLANK	PSH A	
732C	86	2D		LDA A	£'-	GET A '-'
732E	BD	70	E7	JSR	OUT	OUTPUT IT
7331	32			PUL A		
7332	39			RTS		

* GETCCR GET NEW CCR VALUE

7333	CE	77	88	GETCCR	LDX	£FLAGS	POINT TO FLAGS
7336	7F	00	12		CLR	TEMP1	SET BITS 6 AND 7 TO '1'
7339	0D				SEC		
733A	79	00	12		ROL	TEMP1	
733D	0D				SEC		
733E	79	00	12		ROL	TEMP1	
7341	BD	70	DA	NXTCC	JSR	IN	GET A CHARACTER
7344	A1	00			CMP A	0,X	IS IT THE CORRECT FLAG?
7346	26	03			BNE	ZEROCC	NO, TRY A DASH
7348	0D				SEC		YES, SET IT
7349	20	06			BRA	NXTCC1	DO ANOTHER
734B	81	2D		ZEROCC	CMP A	£'-	IS IT A DASH?
734D	27	01			BEQ	NXTCC0	
734F	39				RTS		NO, EXIT FROM THIS MODE
7350	0C			NXTCC0	CLC		YES, CLEAR THE FLAG
7351	79	00	12	NXTCC1	ROL	TEMP1	MOVE TO THE NEXT FLAG
7354	08				INX		
7355	8C	77	8E		CPX	£FLGEND+1	IS IT THE LAST ONE?
7358	26	E7			BNE	NXTCC	NO, DO ANOTHER
735A	96	12			LDA A	TEMP1	YES, SAVE CCR
735C	97	15			STA A	CCR	
735E	39				RTS		

* REGDIS, THIS OUTPUTS THE CONTENTS OF THE REGISTERS

735F	BD	70	C5	REGDIS	JSR	OUTS	
7362	CE	76	D5		LDX	£MSG6	
7365	BD	70	3F		JSR	PSTRNG	
7368	CE	00	17		LDX	£ACCA	OUTPUT ACCA
736B	BD	70	C3		JSR	OUT2HS	
736E	BD	70	C5		JSR	OUTS	
7371	CE	76	E1		LDX	£MSG7	
7374	BD	70	3F		JSR	PSTRNG	
7377	CE	00	16		LDX	£ACCB	OUTPUT ACCB
737A	BD	70	C3		JSR	OUT2HS	
737D	BD	70	C5		JSR	OUTS	
7380	CE	77	08		LDX	£MSG11	
7383	BD	70	3F		JSR	PSTRNG	
7386	CE	00	15		LDX	£CCR	OUTPUT CCR
7389	BD	73	0C		JSR	DISCCR	
738C	BD	74	02		JSR	PCRLF	
738F	C6	04			LDA B	£4	

7391	8D	38		BSR	NXTSPC	
7393	CE	76	ED	LDX	£MMSG8	
7396	BD	70	3F	JSR	PSTRNG	
7399	CE	00	18	LDX	£XH	OUTPUT X REG
739C	BD	70	C1	JSR	OUT4HS	
739F	C6	02		LDA B	£2	
73A1	8D	28		BSR	NXTSPC	
73A3	CE	76	FF	LDX	£MMSG10	
73A6	BD	70	3F	JSR	PSTRNG	
73A9	CE	00	1E	LDX	£SPH2	OUTPUT SP
73AC	BD	70	C1	JSR	OUT4HS	
73AF	C6	02		LDA B	£2	
73B1	8D	18		BSR	NXTSPC	
73B3	CE	76	F6	LDX	£MMSG9	
73B6	BD	70	3F	JSR	PSTRNG	
73B9	CE	00	1A	LDX	£PCH	OUTPUT PC
73BC	BD	70	C1	JSR	OUT4HS	
73BF	7D	00	26	TST	TRFLAG	
73C2	27	01		BEQ	FINREG	
73C4	39			RTS		
73C5	8E	00	7F	FINREG	LDS	£STACK
73C8	7E	71	1B		JMP	CONTRL

* OUTPUT X SPACES, WHERE X IS THE CONTENTS OF ACCB

73CB	BD	70	C5	NXTSPC	JSR	OUTS
73CE	5A				DEC B	
73CF	26	FA			BNE	NXTSPC
73D1	39				RTS	

* INSERT BREAK-POINT

73D2	BD	70	0F	BREAK	JSR	GETADD	GET BP ADDRESS
73D5	A6	00			LDA A	0,X	
73D7	C6	3F			LDA B	£\$3F	
73D9	E7	00			STA B	0,X	PLACE IN A SWI
73DB	CE	00	20		LDX	£BRPTST	POINT TO BP STORE
73DE	A7	00			STA A	0,X	SAVE BP OPCODE
73E0	96	08			LDA A	XTEMP1	SAVE OPCODE ADDRESS
73E2	A7	01			STA A	1,X	
73E4	96	09			LDA A	XTEMP1+1	
73E6	A7	02			STA A	2,X	
73E8	7C	00	2C		INC	BPFLAG	SET BP FLAG
73EB	39				RTS		

* DELETE BREAK-POINT

73EC	CE	00	20	DELBP	LDX	£BRPTST	POINT TO BP STORAGE
73EF	A6	00			LDA A	0,X	GET ORIGINAL BYTE
73F1	EE	01			LDX	1,X	POINT TO IT'S ADDRESS
73F3	A7	00			STA A	0,X	SAVE IT
73F5	7F	00	20		CLR	BRPTST	CLEAR STORAGE
73F8	7F	00	21		CLR	BRPTST+1	
73FB	7F	00	22		CLR	BRPTST+2	
73FE	7F	00	2C		CLR	BPFLAG	CLEAR BP FLAG

7401 39

RTS

* PCRLF, PRINT A CR AND LF

7402	DF	0A	PCRLF	STX	XTEMP2
7404	CE	76 B3		LDX	£MMSG2
7407	BD	70 3F		JSR	PSTRNG
740A	DE	0A		LDX	XTEMP2
740C	39			RTS	

* TRACE

740D	BD	70 0F	TRACE	JSR	GETADD	GET START ADDRESS
7410	8D	F0		BSR	PCRLF	OUTPUT A CR AND LF
7412	96	08		LDA A	XTEMP1	SET PCH FOR AN RTI START
7414	97	1A		STA A	PCH	
7416	96	09		LDA A	XTEMP1+1	
7418	97	1B		STA A	PCH+1	
741A	7C	00 26		INC	TRFLAG	SET TRACE FLAG
741D	CE	75 3F		LDX	£RETSWI	ENSURE SWI VECTOR IS CORRECT
7420	DF	02		STX	SWIV	
7422	BD	71 F7		JSR	UPDAS2	UPDATE SPH
7425	20	06		BRA	TRACEA	
7427	BD	75 8A	TRACE1	JSR	UPDASP	
742A	BD	73 5F		JSR	REGDIS	DISPLAY REGISTERS
742D	7F	00 27	TRACEA	CLR	BRAFLG	RESET BRANCH FLAG
7430	DE	1A		LDX	PCH	GET PROGRAM PC
7432	8D	CE		BSR	PCRLF	
7434	DF	0C		STX	XTEMP3	
7436	CE	00 0C		LDX	£XTEMP3	
7439	BD	70 C1		JSR	OUT4HS	OUTPUT ADDRESS
743C	DE	0C		LDX	XTEMP3	
743E	B6	00		LDA B	0,X	
7440	BD	70 C3		JSR	OUT2HS	OUTPUT OPCODE
7443	D7	28		STA B	INSTR	
7445	A6	00		LDA A	0,X	CHECK FOR SPECIAL INSTRUCTIONS
7447	97	29		STA A	INSTR+1	
7449	A6	01		LDA A	1,X	
744B	97	2A		STA A	INSTR+2	
744D	C1	8D		CMP B	£\$8D	IS IT A BSR?
744F	27	12		BEQ	BRAINS	
7451	C1	8C		CMP B	£\$8C	IS IT A CPX?
7453	27	1D		BEQ	BYTE3	
7455	C1	8E		CMP B	£\$8E	IS IT AN LDS?
7457	27	19		BEQ	BYTE3	
7459	C1	CE		CMP B	£\$CE	IS IT AN LDX?
745B	27	15		BEQ	BYTE3	
745D	C4	F0		AND B	£\$F0	
745F	C1	20		CMP B	£\$20	IS IT A BRANCH
7461	26	05		BNE	NOTBRA	
7463	7C	00 27	BRAINS	INC	BRAFLG	
7466	20	0F		BRA	BYTE2	
7468	C1	60	NOTBRA	CMP B	£\$60	IF LESS THAN THEN SINGLE BYTE
746A	25	0E		BCS	BYTE1	
746C	C4	30		AND B	£\$30	

746E	C1	30		CMP B	££30	
7470	26	05		BNE	BYTE2	
7472	BD	70	C1	JSR	OUT4HS	3 BYTE INSTRUCTION
7475	20	03		BRA	BYTE1	
7477	BD	70	C3	JSR	OUT2HS	2 BYTE INSTRUCTION
747A	DF	0C		STX	XTEMP3	SINGLE BYTE INSTRUCTION
747C	7D	00	27	TST	BRAFLG	IS IT A BRANCH?
747F	27	15		BEQ	NOTBRI	
7481	4F			CLR A		
7482	D6	29		LDA B	INSTR+1	DETERMINE THE BRANCH
7484	2C	02		BGE	BDIR	DIRECTION
7486	86	FF		LDA A	££FF	
7488	DB	0D		ADD B	XTEMP3+1	
748A	99	0C		ADC A	XTEMP3	
748C	97	23		STA A	BRPTST+3	
748E	D7	24		STA B	BRPTST+4	
7490	CE	00	23	LDX	£BRPTST+3	
7493	BD	70	C1	JSR	OUT4HS	DISPLAY DESTINATION ADDR
7496	BD	74	02	JSR	PCRLF	
7499	BD	70	DA	JSR	IN	GET COMMAND
749C	81	20		CMP A	£FORWRD	NEXT INSTRUCTION?
749E	27	18		BEQ	TRACE2	
74A0	CE	77	4F	LDX	£FUTABL	NO, POINT COMMAND TABLE
74A3	A1	00		CMP A	0,X	IS IT A COMMAND?
74A5	27	0B		BEQ	TMATCH	YES, GO AND DO IT
74A7	08			INX		
74A8	08			INX		
74A9	08			INX		
74AA	8C	77	61	CPX	£CTLEND	END OF TABLE?
74AD	26	F4		BNE	TRCON1	NO, TRY ANOTHER
74AF	7E	70	8C	JMP	WHAT1	YES, OUTPUT 'WHAT?'
74B2	EE	01		LDX	1,X	DO COMMAND
74B4	AD	00		JSR	0,X	
74B6	20	E1		BRA	TRCON	GET ANOTHER COMMAND
74B8	C6	3F		LDA B	££3F	CONTINUE TRACE
74BA	96	28		LDA A	INSTR	
74BC	81	8D		CMP A	££8D	IS IT A BSR?
74BE	26	09		BNE	BRATST	
74C0	DE	23		LDX	BRPTST+3	
74C2	DF	0C		STX	XTEMP3	
74C4	7F	00	27	CLR	BRAFLG	CLEAR BRANCH FLAG
74C7	20	1E		BRA	DOINST	DO THE INSTRUCTION
74C9	7D	00	27	TST	BRAFLG	
74CC	27	0A		BEQ	SPCINS	IF ZERO THEN SPECIAL
74CE	DE	23		LDX	BRPTST+3	
74D0	A6	00		LDA A	0,X	
74D2	97	25		STA A	BRPTST+5	
74D4	E7	00		STA B	0,X	
74D6	20	0F		BRA	DOINST	
74D8	CE	75	26	LDX	£OPTAB	POINT TO SPECIAL
74DB	A1	00		CMP A	0,X	INSTRUCTION TABLE
74DD	27	17		BEQ	OPMAT	IS IT SPECIAL?
74DF	08			INX		
74E0	08			INX		
74E1	08			INX		

74E2 8C 75 3E		CPX	£OPTABN	END OF TABLE?
74E5 26 F4		BNE	OPCOM	NO, THEN TRY AGAIN
74E7 DE 0C	DOINST	LDX	XTEMP3	GO AND PROCESS INST.
74E9 A6 00		LDA A	0,X	
74EB 97 2B		STA A	OPCODE	
74ED E7 00		STA B	0,X	
74EF E1 00		CMP B	0,X	
74F1 26 09		BNE	MEMMES	
74F3 7E 72 07		JMP	GORTI	
74F6 EE 01	OPMAT	LDX	1,X	PROCESS SPECIAL INST.
74F8 AD 00		JSR	0,X	
74FA 20 EB		BRA	DOINST	
74FC CE 76 BA	MEMMES	LDX	£MESG4	MEMORY R/W ERROR
74FF BD 70 3F		JSR	PSTRNG	
7502 7E 71 1B		JMP	CONTRL	
7505 DE 29	NORJMP	LDX	INSTR+1	JUMP INSTRUCTION
7507 DF 0C		STX	XTEMP3	
7509 39		RTS		
750A DE 1C	JMPIND	LDX	SPH	INDEXED JUMP
750C A6 05		LDA A	5,X	INSTRUCTION
750E 9B 29		ADD A	INSTR+1	
7510 97 0D		STA A	XTEMP3+1	
7512 A6 04		LDA A	4,X	
7514 89 00		ADC A	£0	
7516 97 0C		STA A	XTEMP3	
7518 39		RTS		
7519 DE 1C	DORTS	LDX	SPH	RTS INST.
751B EE 08		LDX	8,X	
751D 20 04		BRA	SAVERT	
751F DE 1C	DORTI	LDX	SPH	RTI INST.
7521 EE 0D		LDX	13,X	
7523 DF 0C	SAVERT	STX	XTEMP3	
7525 39		RTS		
* OPCODE TABLE				
7526 39	OPTAB	FCB	\$39	
7527 75 19		FDB	DORTS	
7529 3B		FCB	\$3B	
752A 75 1F		FDB	DORTI	
752C 3E		FCB	\$3E	
752D 71 1B		FDB	CONTRL	
752F 3F		FCB	\$3F	
7530 71 1B		FDB	CONTRL	
7532 6E		FCB	\$6E	
7533 75 0A		FDB	JMPIND	
7535 7E		FCB	\$7E	
7536 75 05		FDB	NORJMP	
7538 AD		FCB	\$AD	

7539	75	0A		FDB	JMPIND
753B	BD			FCB	\$BD
753C	75	05		FDB	NORJMP
753E	00		OPTABN	FCB	0

* RETURN HERE FROM SWI

753F	9F	1C	RETSWI	STS	SPH	SAVE SP
7541	30			TSX		
7542	A6	00		LDA A	0,X	SAVE ALL REGISTERS
7544	97	15		STA A	CCR	SAVE CCR
7546	A6	01		LDA A	1,X	
7548	97	16		STA A	ACCB	SAVE ACCB
754A	A6	02		LDA A	2,X	
754C	97	17		STA A	ACCA	SAVE ACCA
754E	A6	03		LDA A	3,X	
7550	97	18		STA A	XH	SAVE X
7552	A6	04		LDA A	4,X	
7554	97	19		STA A	XH+1	
7556	A6	05		LDA A	5,X	
7558	97	1A		STA A	PCH	SAVE PC
755A	A6	06		LDA A	6,X	
755C	97	1B		STA A	PCH+1	
755E	DE	1A		LDX	PCH	ADJUST PC
7560	09			DEX		
7561	DF	1A		STX	PCH	
7563	7D	00	2C	TST	BPFLAG	BP FLAG SET?
7566	26	19		BNE	REGIS	
7568	7D	00	26	TST	TRFLAG	ARE WE TRACING?
756B	27	14		BEQ	REGIS	
756D	DE	0C		LDX	XTEMP3	
756F	96	2B		LDA A	OPCODE	
7571	A7	00		STA A	0,X	
7573	7D	00	27	TST	BRAFLG	
7576	27	06		BEQ	NEXT	
7578	DE	23		LDX	BRPTST+3	
757A	96	25		LDA A	BRPTST+5	
757C	A7	00		STA A	0,X	
757E	7E	74	27	NEXT	JMP	TRACE1
7581	BD	75	8A	REGIS	JSR	UPDASP
7584	BD	73	5F		JSR	REGDIS
7587	7E	71	1B		JMP	CONTRL

* UP-DATE SP

758A	CE	00	1C	UPDASP	LDX	\$SPH
758D	A6	01			LDA A	1,X
758F	8B	07			ADD A	\$7
7591	A7	03			STA A	3,X
7593	A6	00			LDA A	0,X
7595	89	00			ADC A	\$0
7597	A7	02			STA A	2,X
7599	39				RTS	

* LOAD DATA FROM CASSETTE

759A	86	55	LOAD	LDA A	£\$55	SET UP ACIA
759C	B7	40 00		STA A	ACIAC	
759F	7C	00 2D		INC	LDFLG	SET THE LOAD FLAG
75A2	BD	70 DA	LOAD1	JSR	IN	GET A CHARACTER
75A5	81	53		CMP A	£'S	IS IT AN 'S'?
75A7	26	F9		BNE	LOAD1	WAIT TILL AN 'S' IS FOUND
75A9	BD	70 DA		JSR	IN	GET ANOTHER CHARACTER
75AC	81	39		CMP A	£'9	IS IT A '9'?
75AE	27	29		BEQ	LOAD5	
75B0	81	31		CMP A	£'1	IS IT A '1'?
75B2	26	EE		BNE	LOAD1	NO - WAIT FOR NEXT 'S'
75B4	7F	00 13		CLR	CHKSUM	SET CHECKSUM TO ZERO
75B7	BD	70 1A		JSR	GETBYT	GET A BYTE
75BA	80	02		SUB A	£2	
75BC	97	14		STA A	BYTNUM	NUMBER OF BYTES IN LINE
75BE	BD	70 0F		JSR	GETADD	GET THE START ADDRESS
75C1	BD	70 1A	LOAD2	JSR	GETBYT	
75C4	7A	00 14		DEC	BYTNUM	DECREMENT THE BYTE COUNTER
75C7	27	05		BEQ	LOAD3	
75C9	A7	00		STA A	0,X	
75CB	08			INX		
75CC	20	F3		BRA	LOAD2	
75CE	7C	00 13	LOAD3	INC	CHKSUM	
75D1	27	CF		BEQ	LOAD1	
75D3	CE	77 3C		LDX	£MMSG19	
75D6	BD	70 3F		JSR	PSTRNG	
75D9	7F	00 2D	LOAD5	CLR	LDFLG	
75DC	39			RTS		

* WRITE TO THE CASSETTE

75DD	BD	72 3A	WRITE	JSR	LIMITS	
75E0	86	15		LDA A	£\$15	SET UP ACIA
75E2	B7	40 00		STA A	ACIAC	
75E5	CE	00 78	HEADER	LDX	£120	OUTPUT A 4 SECOND HEADER
75E8	86	55	HEADE1	LDA A	£'U	OF 'U'
75EA	BD	70 E7		JSR	OUT	
75ED	09			DEX		
75EE	26	F8		BNE	HEADE1	
75F0	96	11	WRITE1	LDA A	ENDADD+1	START WRITE TO TAPE
75F2	90	0F		SUB A	STARAD+1	
75F4	D6	10		LDA B	ENDADD	
75F6	D2	0E		SBC B	STARAD	
75F8	26	04		BNE	WRITE2	
75FA	81	10		CMP A	£16	
75FC	25	02		BCS	WRITE3	
75FE	86	0F	WRITE2	LDA A	£15	
7600	8B	04	WRITE3	ADD A	£4	
7602	97	14		STA A	BYTNUM	FRAME COUNT
7604	80	03		SUB A	£3	
7606	97	12		STA A	TEMP1	BYTE COUNT
7608	CE	77 49		LDX	£MMSG20	
760B	BD	70 3F		JSR	PSTRNG	

760E 5F		CLR B		CLEAR CHECKSUM
760F CE 00 14		LDX	£BYTNUM	
7612 8D 2A		BSR	OUTTAP	OUTPUT FRAME COUNT
7614 CE 00 0E		LDX	£STARAD	
7617 8D 25		BSR	OUTTAP	OUTPUT ADDRESS
7619 8D 23		BSR	OUTTAP	
761B DE 0E		LDX	STARAD	
761D 8D 1F	WRITE4	BSR	OUTTAP	OUTPUT A BYTE
761F 7A 00 12		DEC	TEMP1	DEC BYTE COUNT
7622 26 F9		BNE	WRITE4	
7624 DF 0E		STX	STARAD	
7626 53		COM B		
7627 37		PSH B		
7628 30		TSX		
7629 8D 13		BSR	OUTTAP	OUTPUT CHECKSUM
762B 33		PUL B		
762C DE 0E		LDX	STARAD	
762E 09		DEX		
762F 9C 10		CPX	ENDADD	
7631 26 BD		BNE	WRITE1	END? THEN OP AN 'S9'
7633 86 53		LDA A	£'S	'S'
7635 BD 70 E7		JSR	OUT	
7638 86 39		LDA A	£'9	'9'
763A BD 70 E7		JSR	OUT	
763D 39		RTS		
763E EB 00	OUTTAP	ADD B	0,X	
7640 7E 70 B6		JMP	OUT2H	

* 8 BIT DIGITAL TO ANALOG OUTPUT ROUTINE

7643 B7 3F 00	DAC8	STA A	DACDAT	ON EXIT ACC A WILL
7646 7F 3F 01		CLR	DACDAT+1	BE UNCHANGED
7649 39		RTS		

* 10 BIT DIGITAL TO ANALOG OUTPUT ROUTINE

764A DF 06	DAC10	STX	XTEMP	ON ENTRY THE DATA
764C 36		PSH A		SHOULD BE IN THE X REG
764D 37		PSH B		
764E 96 07		LDA A	XTEMP+1	
7650 D6 06		LDA B	XTEMP	
7652 C4 03		AND B	£3	
7654 48		ASL A		DATA IS ARRANGED BY
7655 59		ROL B		THIS ROUTINE SO THAT
7656 48		ASL A		IT IS IN THE FORMAT
7657 59		ROL B		REQUIRED BY THE DIGITAL
7658 48		ASL A		TO ANALOG CONVERTER
7659 59		ROL B		
765A 48		ASL A		
765B 59		ROL B		
765C 48		ASL A		
765D 59		ROL B		
765E 48		ASL A		
765F 59		ROL B		

7660	F7	3F	00	STA	B	DACDAT
7663	B7	3F	01	STA	A	DACDAT+1
7666	33			PUL	B	
7667	32			PUL	A	
7668	39			RTS		

* MESSAGE TABLE

7669	20	MESG0	FCC	"	0	1	2	3	4	5	6	7	"
766A	20 20												
766C	20 20												
766E	20 20												
7670	30 20												
7672	20 31												
7674	20 20												
7676	32 20												
7678	20 33												
767A	20 20												
767C	20 34												
767E	20 20												
7680	35 20												
7682	20 36												
7684	20 20												
7686	37 20												
7688	20												
7689	20		FCC	" 8 9 A B C D E F "									
768A	38 20												
768C	20 39												
768E	20 20												
7690	41 20												
7692	20 42												
7694	20 20												
7696	20 43												
7698	20 20												
769A	44 20												
769C	20 45												
769E	20 20												
76A0	46 20												
76A2	20 20												
76A4	20 20												
76A6	20 20												
76A8	20												
76A9	41		FCC	"ASCII"									
76AA	53 43												
76AC	49 49												
76AE	04		FCB	EOT									
76AF	0A	MESG1	FCB	LF,CR,PROMPT,EOT									
76B0	0D 3E												
76B2	04												
76B3	0A	MESG2	FCB	LF,CR,EOT									
76B4	0D 04												
76B6	0A	MESG3	FCB	LF,CR									
76B7	0D												
76B8	20		FCC	" "									
76B9	04		FCB	EOT									

76BA 4E	MESG4	FCC	"NOT R/W MEMORY!"
76BB 4F 54			
76BD 20 52			
76BF 2F 57			
76C1 20 4D			
76C3 45 4D			
76C5 4F 52			
76C7 59 21			
76C9 04		FCB	EOT
76CA 0A	MESG5	FCB	LF,CR
76CB 0D			
76CC 57		FCC	"WHAT?"
76CD 48 41			
76CF 54 3F			
76D1 04		FCB	EOT
76D2 5E	MESG6A	FCC	"^A "
76D3 41 20			
76D5 41	MESG6	FCC	"ACC A = "
76D6 43 43			
76D8 20 41			
76DA 20 3D			
76DC 20			
76DD 04		FCB	EOT
76DE 5E	MESG7A	FCC	"^B "
76DF 42 20			
76E1 41	MESG7	FCC	"ACC B = "
76E2 43 43			
76E4 20 42			
76E6 20 3D			
76E8 20			
76E9 04		FCB	EOT
76EA 5E	MESG8A	FCC	"^X "
76EB 58 20			
76ED 49	MESG8	FCC	"IX = "
76EE 58 20			
76F0 3D 20			
76F2 04		FCB	EOT
76F3 5E	MESG9A	FCC	"^P "
76F4 50 20			
76F6 50	MESG9	FCC	"PC = "
76F7 43 20			
76F9 3D 20			
76FB 04		FCB	EOT
76FC 5E	MESG0A	FCC	"^S "
76FD 53 20			
76FF 53	MESG10	FCC	"SP = "
7700 50 20			
7702 3D 20			
7704 04		FCB	EOT
7705 5E	MESG1A	FCC	"^C "
7706 43 20			
7708 43	MESG11	FCC	"CCR = "
7709 43 52			
770B 20 3D			
770D 20			

770E 04		FCB	EOT
770F 0A	MESG13	FCB	LF,LF,LF,LF,CR
7710 0A 0A			
7712 0A 0D			
7714 20		FCC	" -- DATERM --"
7715 20 20			
7717 2D 2D			
7719 20 44			
771B 41 54			
771D 45 52			
771F 4D 20			
7721 2D 2D			
7723 0A		FCB	LF,CR
7724 0D			
7725 20		FCC	" V1.3 "
7726 20 20			
7728 20 20			
772A 20 20			
772C 56 31			
772E 2E 33			
7730 20 20			
7732 20			
7733 0A		FCB	LF,LF,LF,CR,EOT
7734 0A 0A			
7736 0D 04			
7738 0A	MESG18	FCB	LF,CR
7739 0D			
773A 3F		FCC	"?"
773B 04		FCB	EOT
773C 0A	MESG19	FCB	LF,CR
773D 0D			
773E 4C		FCC	"LOAD ERROR"
773F 4F 41			
7741 44 20			
7743 45 52			
7745 52 4F			
7747 52			
7748 04		FCB	EOT
7749 0D	MESG20	FCB	CR,LF,0
774A 0A 00			
774C 53		FCC	"S1"
774D 31			
774E 04		FCB	EOT
774F 01	FUTABL	FCB	'A'-CNTRL DISPLAY/CHANGE ACCA
7750 72 A5		FDB	ADPCN
7752 02		FCB	'B'-CNTRL DISPLAY/CHANGE ACCB
7753 72 B0		FDB	BDPCN
7755 03		FCB	'C'-CNTRL DISPLAY/CHANGE CCR
7756 72 BB		FDB	CDPCN
7758 0D		FCB	'M'-CNTRL GOTO CONTROL
7759 71 1B		FDB	CONTRL
775B 10		FCB	'P'-CNTRL DISPLAY/CHANGE PC
775C 72 CB		FDB	PDPCN

775E 18		FDB	'X'-CNTRL DISPLAY/CHANGE IX
775F 72 E1		FDB	XDPCN
7761 13	CTLEND	FDB	'S'-CNTRL DISPLAY/CHANGE SP
7762 72 D6		FDB	SDPCN

7764 42		FCC	"B" SET BREAK-POINTS
7765 73 D2		FDB	BREAK
7767 44		FCC	"D" PRINT CONTENTS OF MEMORY
7768 71 69		FDB	DUMP
776A 46		FCC	"F" FILL MEMORY WITH A BYTE
776B 72 2A		FDB	FILL
776D 47		FCC	"G" GO AND EXECUTE PROGRAM
776E 72 07		FDB	GORTI
7770 49		FCC	"I" INSPECT MEMORY LOCATION
7771 70 46		FDB	INSPEC
7773 4A		FCC	"J" JUMP TO ADDRESS
7774 71 F2		FDB	JUMP
7776 4C		FCC	"L" LOAD FROM CASSETTE
7777 75 9A		FDB	LOAD
7779 4D		FCC	"M" BLOCK MOVE MEMORY
777A 72 4B		FDB	MOVE
777C 52		FCC	"R" DISPLAY REGISTERS
777D 73 5F		FDB	REGDIS
777F 54		FCC	"T" TRACE USER'S PROGRAM
7780 74 0D		FDB	TRACE
7782 57		FCC	"W" WRITE TO CASSETTE
7783 75 DD		FDB	WRITE
7785 58		FCC	"X" DELETE BREAK-POINT
7786 73 EC		FDB	DELBP
7788	TABLEN EQU		*

* CONDITION CODE TABLE

7788 48	FLAGS	FCC	"H"
7789 49		FCC	"I"
778A 4E		FCC	"N"
778B 5A		FCC	"Z"
778C 56		FCC	"V"
778D 43	FLGEND	FCC	"C"
77F8		ORG	\$77F8
77F8 70 0B		FDB	IRQ
77FA 70 00		FDB	SWI
77FC 70 04		FDB	NMI
77FE 70 08		FDB	RESET
		END	

NO ERROR(S) DETECTED

SYMBOL TABLE:

ACCA 0017	ACCB 0016	ACIAC 4000	ACIAD 4001	ADPCN 72A5
ASCII 0027	ASCII 71B6	BDIR 7488	BDPCN 72B0	BLANK 732B
BLOCHI 7287	BPFLAG 002C	BRAFLG 0027	BRAINS 7463	BRATST 74C9

BREAK	73D2	BRPTST	0020	BYTE1	747A	BYTE2	7477	BYTE3	7472
BYTNUM	0014	CCR	0015	CDPCN	72BB	CHKEND	71CA	CHKSUM	0013
CNTRL	0040	CONTRL	711B	CR	000D	CTLCHR	714C	CTLEND	7761
DAC10	764A	DAC8	7643	DACDAT	3F00	DECLP	7273	DELBP	73EC
DISCCR	730C	DOINST	74E7	DORTI	751F	DORTS	7519	DOT	71C5
DPCN	72EC	DPCN1	72F9	DUMP	7169	ENDADD	0010	EOT	0004
ESC	001B	EXIT	71EF	FILL	722A	FILL1	7232	FINREG	73C5
FLAGS	7788	FLGEND	778D	FORWRD	0020	FUTABL	774F	GETADD	700F
GETBY2	701C	GETBYT	701A	GETCCR	7333	GETH2	70A3	GETH3	70B5
GETHEX	7097	GORTI	7207	HANGUP	71E5	HEADE1	75E8	HEADER	75E5
IN	70DA	IN1	70DB	INCLP	728C	INCPTR	7321	INIT	710A
INSP5	704B	INSP6	707B	INSP7	7075	INSP8	708F	INSP9	7092
INSPEC	7046	INSTR	0028	IRQ	700B	IRQV	0004	JMPIND	750A
JUMP	71F2	LAST	72A0	LDFLG	002D	LF	000A	LIMITS	723A
LOAD	759A	LOAD1	75A2	LOAD2	75C1	LOAD3	75CE	LOAD5	75D9
MATCH	7163	MEMMES	74FC	MESGO	7669	MESGOA	76FC	MSG1	76AF
MESG10	76FF	MESG11	7708	MESG13	770F	MESG18	7738	MESG19	773C
MESG1A	7705	MESG2	76B3	MESG20	7749	MESG3	76B6	MESG4	76BA
MESG5	76CA	MESG6	76D5	MESG6A	76D2	MESG7	76E1	MESG7A	76DE
MESG8	76ED	MESG8A	76EA	MESG9	76F6	MESG9A	76F3	MOVE	724B
NEXBYT	71A3	NEXLIN	7197	NEXPA1	7188	NEXPAG	7185	NEXT	757E
NMI	7004	NMIV	0000	NOINIT	7135	NORJMP	7505	NOT2	71AF
NOTBRA	7468	NOTBRI	7496	NXTCC	7341	NXTCC0	7350	NXTCC1	7351
NXTCHR	714F	NXTFLG	7313	NXTSPC	73CB	OPCODE	002B	OPCOM	74DB
OPMAT	74F6	OPTAB	7526	OPTABN	753E	OUT	70E7	OUT1	70E8
OUT2H	70B6	OUT2HS	70C3	OUT4HS	70C1	OUTHL	702A	OUTHR	702E
OUTHR1	7038	OUTS	70C5	OUTTAP	763E	PAUSE	70CA	PAUSE2	70D3
PAUSE3	70D8	PCH	001A	PCRLF	7402	PDPCN	72CB	PROMPT	003E
PSTRN0	703B	PSTRNG	703F	REGDIS	735F	REGIS	7581	RESET	7008
RETSWI	753F	REVERS	005E	SAVERT	7523	SDPCN	72D6	SETBIT	731A
SPCINS	74D8	SPH	001C	SPH2	001E	STACK	007F	STARAD	000E
START1	70F4	SWI	7000	SWIV	0002	TABLEN	7788	TEMP1	0012
TEST10	71E1	TMATCH	74B2	TRACE	740D	TRACE1	7427	TRACE2	74B8
TRACEA	742D	TRCON	7499	TRCON1	74A3	TRFLAG	0026	UPDAS2	71F7
UPDASP	758A	USTACK	0060	WHAT	715B	WHAT1	708C	WRITE	75DD
WRITE1	75F0	WRITE2	75FE	WRITE3	7600	WRITE4	761D	XDPCN	72E1
XH	0018	XTEMP	0006	XTEMP1	0008	XTEMP2	000A	XTEMP3	000C
ZEROCC	734B								

SECTION 2

USERS MANUAL FOR THE BASIC INTERPRETER

by S GÖGLER

CONTENTS

	Page
INTRODUCTION	1
HARDWARE REQUIRED	2
STARTING UP	3
BASIC INTERPRETER OPERATION	4
1. INTRODUCTION	4
2. THE BASIC OPERATING SYSTEM	5
2.1 Operating System Commands	5
LIST, MON, NEW, RUN	
3. IMMEDIATE MODE	7
4. THE EDITOR	8
4.1 Inserting a Program Line	8
4.2 Deleting a Program Line	9
4.3 Changing a Program Line	10
5. BASIC LANGUAGE FEATURES	11
5.1 Variables in BASIC	11
5.2 BASIC Expressions	12
Arithmetic Operators, Logical	
Operators, Relational Operators	
5.3 BASIC Functions	14
PEEK	
5.4 BASIC Statements	15
DATA, END, FOR..TO..STEP, GOSUB,	
GOTO, IF..THEN, INPUT, LET, NEXT,	
POKE, PRINT, READ, REM, RETURN, STOP	
6. ERROR MESSAGES	20
 <u>APPENDICES :</u>	
A/ DATUM INTEGER BASIC SUMMARY	21
B/ DATUM INTEGER BASIC ERROR CODES	23
C/ ASCII CHARACTER SET	25
D/ SAMPLE RUN	27
E/ GLOSSARY	29

INTRODUCTION

The BASIC Interpreter included in this package is only a simple example of the more complex commercially available interpreters usually found in larger micro computers. This version is intended primarily as a teaching aid to be used in conjunction with the DATUM computer kit, developed by the South Australian Institute of Technology.

The purpose of this manual is to introduce the user to DATUM Integer BASIC. In the following pages information on the hardware required, the Interpreter start-up procedures, and the general operation of Integer BASIC are presented.

HARDWARE REQUIRED

The equipment you will need to use the DATUM integer BASIC Interpreter is listed :

- * DATUM Microcomputer kit
- * DATUM Extension
- * Full character keyboard for entering programs and data into the computer
- * Display screen for displaying results and other information

DATUM Integer BASIC fills approximately 4 Kbytes of EPROM. The amount of RAM available for BASIC programs totals 4 Kbytes.

STARTING UP

For the setting up and connection of the hardware, and also for the power up procedures please consult your DATUM Computer kit and Extension kit manuals.

It is assumed now that you have in front of you the necessary hardware set up and turned on.

Firstly, before running BASIC, the monitor program must be started. This is achieved by pressing the black 'RS' (Reset) key located to the right of the hexpad on the DATUM kit. The monitor program should now execute, producing the following startup message and prompt on the display screen :

```
-- DATERM --  
V1.3
```

>

You are now ready to run BASIC.

There are two startup addresses possible for DATUM Integer BASIC. The first address at 2800 hex is for cold starts. During a "cold start" the entire user memory is cleared, along with the user variable area. Hence all programs and data that may have been present are lost when BASIC is started from this address.

The other start address, at 2803 hex, is actually a restart address used for "warm starts". A "warm start" is one in which none of the user memory is affected. This restart address thus provides the ability to reenter BASIC, without affecting the current program or data in memory. This means that control can be passed back and forth between BASIC and the monitor program, allowing you to examine what is happening to your BASIC program at the machine code level.

Now, to begin a session of BASIC, use the keyboard to type J (for jump) then the required start address for BASIC :

```
J 2800    for start  
J 2803    for restart
```

Note that a space is automatically inserted by the monitor program between the J and the first digit of the address on the screen.

You should now be confronted by BASIC's startup message which looks like this :

```
<BASIC>
```

!

The "!" character is BASIC's prompt at which you may enter any BASIC instruction. In BASIC everything that is typed must be followed by a <RETURN> to let the interpreter know that your instruction is ready for processing.

At this stage you are ready to use DATUM's Integer BASIC.

BASIC INTERPRETER OPERATION

1. INTRODUCTION

There are so many concepts within BASIC that obviously it would take a long time to describe them all. This manual is not intended to teach BASIC or to show all the concepts, but instead it describes the facilities available within this version of BASIC. The skill of programming in BASIC is left for you to learn.

2. THE BASIC OPERATING SYSTEM

As the central point of the BASIC interpreter, a simple operating system is provided to control and direct all processes. The operating system may be characterised by the prompt : "!" , which implies that it is ready to accept a line of input. A number of rules apply during entry of a line of input :

- a. The limit on the length of an input line is 72 characters. If this limit is exceeded then the line is accepted automatically (without waiting for <RETURN> to be pressed) and processed as it is.
- b. Only upper case alphabetic characters are permitted - this applies to all modes of operation !
- c. Backspace is available by typing <CTRL><H> at the same time.
- d. Pressing <BREAK> at any time while entering a line of input will cancel that line and result in the message :

STOPPED AT 0000

followed by a new prompt.

2.1 Operating System Commands

There are four operating system commands used to control the system. Descriptions of each follow :

2.1.1 LIST

Displays the program currently in memory in sequential order of line number. If the command is followed by a single line number then only the line defined by that line number is displayed. If two line numbers follow the command then all lines between those two lines inclusive are listed on the screen.

examples :

LIST	lists the entire program
LIST 10	lists line 10 only (if it exists)
LIST 10 50	lists all lines between 10 and 50 inclusive (if they exist)

The <BREAK> key may be used to halt a listing. If this is used the message :

STOPPED AT 0000

is displayed to indicate that the listing was halted.

2.1.2 MON

Returns control to the monitor program.

example :

MON

2.1.3 NEW

Clears the user program area and resets all variables to zero.

example :

NEW

2.1.4 RUN

First sets all variables to zero, then executes the BASIC program currently in memory beginning at the smallest line number found. Pressing <BREAK> halts program execution as soon as the line being executed at the time has completed its processing. This also results in the message :

STOPPED AT aaaa

where aaaa was the line number being processed when <BREAK> was pressed.

example :

RUN

To execute a program from a line number other than the first use GOTO. This does not clear the variables.

Rules for operating system commands :

- a. All commands are distinguished by their first three characters only.
- b. LIST must be followed by at least one space if it contains line number(s) as operand(s).
- c. Spaces may be used freely preceding commands, following commands and between commands and their operands.
- d. Operating system commands cannot be placed within the user program. That is, they cannot be preceded by a line number.
- e. If there is an error in an operating system command or if something of no sense at all is entered then you are informed of this via the message :

WHAT?

3. IMMEDIATE MODE

Immediate mode, otherwise known as direct mode, is the situation where BASIC commands are entered directly into the interpreter, without line numbers, and are executed immediately. This version of BASIC allows a subset of the full list of BASIC statements to be used in immediate mode. The subset is made up of the statements :

END	PRINT
GOTO	REM
IF...THEN	RETURN
LET	STOP
POKE	

, along with any assignment expressions and the function PEEK.

No other statements other than those listed may be used in immediate mode.

In error messages produced due to bad immediate mode operations the line number given will be 0000. This line number is representative of immediate mode, and hence cannot be entered as a program line.

The BASIC statements above operate the same in immediate mode as they do in execution (RUN) mode. Two statements deserve a brief mention here as to their special capabilities :

GOTO begins execution of a program from the line number given, hence can be used to execute a program from a line number other than the first. Also GOTO does not clear the variables.

RETURN can be used to continue program execution if it was stopped by <BREAK> or STOP. An error will result if RETURN is attempted after END is executed or after program execution is complete.

4. THE EDITOR

The editor provides the facility to insert, delete and change BASIC program lines. The editor is invoked automatically on finding the first nonspace character to be numeric in a line of user input. Note that the range of line numbers that can be used is 1 through 9999 inclusive. That is, a maximum of 4 digits will be accepted.

Use of the editor is easy :

4.1 Inserting a Program Line

To insert a line into the program simply type in the line with a new (as yet unused) line number. The line will be inserted into its correct position in sequence by the interpreter, which means that lines may be entered in any order.

It is recommended that line numbers be chosen such that new lines may be inserted between existing lines easily. A good practice is to use multiples of 10 initially.

examples :

(note that in all examples the operating system prompt "!" signifies user input.)

now enter :

```
! NEW <RETURN>
! 10 PRINT "HELLO" <RETURN>
! 20 GOTO 10 <RETURN>
```

to see the program type :

```
! LIST <RETURN>
```

the screen should display :

```
0010 PRINT "HELLO"
0020 GOTO 10
```

to illustrate insertion between lines type :

```
! 15 PRINT <RETURN>
! LIST <RETURN>
```

the new line is inserted between the first two :

```
0010 PRINT "HELLO"
0015 PRINT
0020 GOTO 10
```

4.2 Deleting a Program Line

To remove a line from a BASIC program just type its line number and press <RETURN>. The line is deleted immediately.

examples :

enter the above program into memory again.
now to delete line 15 :

```
! 15 <RETURN>
! LIST <RETURN>
```

```
0010 PRINT "HELLO"
0020 GOTO 10
```

if we attempt to delete line 15 again...

```
! 15 <RETURN>
```

WHAT?

...we notice that the editor has recognised our mistake in attempting to delete a line which doesn't exist, and so the "WHAT?" message is displayed to inform us.

4.3 Changing a Program Line

Changing a program line is done in much the same way as insertion is done. To change a line that already exists just retype that line with the same line number. The old line will be discarded and the new line put in its place.

example :

to change line 10 in the above program all we have
to do is retype it :

```
! 10 PRINT "HI "; <RETURN>  
! LIST <RETURN>
```

```
0010 PRINT "HI ";  
0020 GOTO 10
```

5. BASIC LANGUAGE FEATURES

5.1 Variables in BASIC

DATUM BASIC is integer BASIC only, with the range of integers able to be represented being -32768 to +32767.

The variables available for use are solely the letters of the alphabet : A, B, C, ... , Z. This gives a total of 26 variables, all of which are integer variables. Additional variables cannot be defined or referenced - if attempted an error will result.

5.2 BASIC Expressions

An expression may consist of one or more variables and/or constants linked together using any of the valid operators. There are 3 types of operators :

5.2.1 Arithmetic Operators

+	addition
-	subtraction
*	multiplication
/	integer division

Since this version of BASIC is integer only then division truncates any fractional part of the result. For example :

```
5 / 2 = 2
1 / 5 = 0
101 / 10 = 10
6 / 3 = 2
```

5.2.2 Logical Operators

AND	logical AND
OR	logical OR
NOT	logical NOT

5.2.3 Relational Operators

=	equal to
<	less than
>	greater than

Parentheses are available for the grouping together of sub expressions.

examples :

```
(A + B)/2
2*(A/(B-C))
(I-J)*(K + L)
(I = 0) AND (NOT X)
```

The order of evaluation of complex expressions is from left to right. No operator precedence is considered but parentheses may be used if operator precedence is required. Some examples of expression evaluation follow :

```
5*3 - 1/2 = 7
2*(1 + 4) + 2 * -2 = -24
NOT 0 OR 0 * 8 = 8
```

In DATUM Integer BASIC assignment expressions are special in the way that logical and relational operators may be used. This allows for variables to be used as logical variables, where a value of 0 implies 'false' and any other value implies 'true'.

examples :

assume the values : A = 3, B = 0, C = -3

D = A=B	D is assigned the value 0 ('false')
E = NOT B	E is assigned the value 1 ('true')
F = (B>A) OR C	F is assigned the value 1 ('true')
G = (B<A) + C	G is assigned the value -2 ('true')

This special characteristic also carries over to the IF statement :

examples :

IF A THEN ...	executes 'true' path
IF A+C THEN ...	executes 'false' path
IF (A=C) + B THEN ...	executes 'true' path

5.3 BASIC Functions

One function is supplied which can be used within any expression :

5.3.1 PEEK(address)

Returns the value stored at the specified address (in decimal form). For memory addresses 0 to 32767 the value of address is as normal. For memory addresses above 32767 the value of address can be calculated from the following formula :

address = memory address - 65536

examples :

assume the values : I = 32000, J = 123

A = PEEK(299)	A is assigned the value of memory location 299
B = PEEK(-I)	B is assigned the value of memory location 33536
C = PEEK((I+J)/3)	C is assigned the value of memory location 10707

5.4 BASIC Statements

There are a number of BASIC statements (otherwise known as reserved or key words) that can be used within your BASIC programs for input, output, assignment, control, testing and other functions.

One set of rules governs all BASIC statements :

- Only one statement is allowed per line, with the exception of IF which may have one additional statement following the keyword "THEN".
- All keywords are distinguished by their first three characters only.
- All keywords with operands must be followed by at least one space delimiter.
- Spaces may be used freely preceding keywords, following keywords, and between keywords and their operands.

The complete list of DATUM BASIC statements follows :

5.4.1 DATA datalist

Allows you to store data within your program for access via the READ statement. The datalist must be made up of integer constants separated by commas.

examples :

```
10 DATA 20,-5,1,233,0
20 DATA 5
30 DATA -157,-2000,2,2,3,199,10,-11
```

see also : READ

5.4.2 END

Terminates execution of a BASIC program. END is normally used to terminate program execution at some point other than the physical end of the program.

example :

```
999 END
```

5.4.3 FOR variable=exprsn1 TO exprsn2 STEP exprsn3

Opens an iterative loop and executes all statements between this opening statement and the corresponding close loop statement (NEXT variable). Initially variable is set to the value of exprsn1 and on each pass through the loop variable is incremented (or decremented) by the value in exprsn3. The loop completes when the value of variable is greater than that of exprsn2 (for positive step values), or if variable is less than exprsn2 (for negative step values).

STEP is optional - if omitted then the stepping value is assumed to be +1. The value of exprsn3 may be positive for forward stepping or negative for backward stepping.

Note that the values of the expressions are evaluated at each pass through the loop.

examples :

```
20 FOR A=0 TO 10
15 FOR I=10 TO -10           in this case the loop will
                             execute 0 times

40 FOR J=A+1 TO N STEP -3
25 FOR K=1 TO (Y+2)/5 STEP 0  this will cause an infinite
                             loop
```

see also : NEXT

5.4.4 GOSUB linenummer

Transfers control to a subroutine beginning at the line specified by linenummer. The number of the line immediately following the GOSUB statement is stored so that it can be RETURNed to on completion of the subroutine.

examples :

```
900 GOSUB 10
100 GOSUB 180
```

see also : RETURN

5.4.5 GOTO linenummer

Transfers control to the line specified by linenummer.

examples :

```
10 GOTO 9999
80 GOTO 40
15 GOTO 15           this will cause an infinite loop
```

5.4.6 IF exprsn THEN statement

Tests if the value of expression is 'true' or 'false' and executes the statement following THEN only if it is 'true'. If the value of expression is 'false' then execution continues at the next program line in sequence.

Statement may be any BASIC statement other than DATA, FOR...TO...(STEP), IF, INPUT, NEXT or REM.

examples :

```
28 IF A<B THEN GOTO 100
30 IF X THEN Y = Y + 1
60 IF I+J = (K/2)-1 THEN PRINT "I=";I
```

5.4.7 INPUT variablelist

Halts program execution temporarily until an integer constant is entered for all variables in variablelist.

Variablelist may contain any number of variables as long as each is separated by a comma.

example :

```
50 INPUT A,B,C
```

On execution of INPUT a prompt is displayed to let you know that input is required. The prompt looks like this :

?

You may then enter a value for each variable in the list, each value being separated by a comma. If insufficient values are entered then another prompt will be displayed. If too many values are entered then the extra values are ignored, and if you enter a bad value (an alphabetic character for example) BASIC will tell you via the message :

BAD INPUT

and you will have to reenter values for the entire variable list again.

5.4.8 LET variable = expression

Assigns the value of expression to variable. The LET keyword is optional.

examples :

```
11 LET A = 5
12 LET I = (A+B)*C
60 J = J - 1
55 T = (A>B)
```

5.4.9 NEXT variable

Used to close the FOR loop associated with variable.

example :

```
75 NEXT I
```

see also : FOR...TO...(STEP)

5.4.10 POKE address,expression

Places the evaluated value of expression at the memory byte specified by address. The address value may be a constant, variable or expression. Addressing is carried out in the same way as it is for the function PEEK. That is, for memory addresses 0 to 32767 the value of address is as normal, and for memory addresses above 32767 the value of address can be calculated from the formula :

address = memory address - 65536

A memory byte can hold only the values 0 to 255 (00 to FF hex). If the evaluated value of expression is greater than this then the following operation is performed on the evaluated value to bring it into this range, and the result converted to hex before being stored at the appropriate memory byte :

value = REMAINDER(value/256)

examples :

```
10 POKE '17999,255
20 POKE I - 65536,0
30 POKE -199,X/Y
```

see also : PEEK

BEWARE : POKEing values into random locations can be disastrous. Ensure that you know where in memory you are POKEing. If strange things begin to happen after POKEing then it is advised that you reset the computer or start again from power up rather than continue.

5.4.11 PRINT itemlist

Displays all items in itemlist on the screen. The print items in itemlist may be strings (which must be enclosed in double quotes), constants, variables, and/or expressions.

If itemlist is empty then a blank line is displayed, otherwise all items must be separated either by commas or by semicolons.

If a semicolon is used between two PRINT items then no spacing occurs, however if a comma is used then the cursor advances to the next print zone to print the next item. The print zones each consist of 8 columns and start at columns 1, 9, 17, 25, 33, 41, 49, 57 and 65.

A trailing semicolon or comma suppresses linefeed so that the next PRINT begins on the same line.

Positive integers are printed with a leading space and negative integers are printed with a leading negative sign. All integers are printed with a trailing space.

examples :

```
10 PRINT "HELLO"
15 PRINT
20 PRINT "ANSWER = ",2*X-6,"METRES"
35 PRINT "NEXT PRINT WILL BEGIN ON THIS LINE ";
70 PRINT I,J,, "K=";K
```

5.4.12 READ variablelist

Assigns a value to each variable in variablelist from the DATA statements found in the BASIC program. The variables in variablelist must be separated by commas.

Starting from the very first READ the first value in the first DATA statement is taken, then the second value, then the third, and so on. Once all the values in one DATA statement

have been used, the next READ takes the first value from the next DATA statement found in the BASIC program.

example :

```
30 READ I,J,X
```

see also : DATA

5.4.13 REM usercomments

Instructs the interpreter to ignore the rest of the line. The purpose of this is so that you can place comments or REMarks into your program as documentation.

examples :

```
30 REM THIS LINE IS A COMMENT
35 REM
45 REM THIS SUBROUTINE DOES ...
```

5.4.14 RETURN

Ends a subroutine and returns control to the program line immediately following the most recently executed GOSUB.

example :

```
40 RETURN
```

see also : GOSUB

5.4.15 STOP

Breaks program execution and displays the message :

```
STOPPED AT aaaa
```

where aaaa is the line number of the STOP statement.

The main purpose for the STOP statement is for debugging - once the program stops you can examine or alter the values of variables. RETURN then continues execution again (see Immediate Mode : RETURN).

example :

```
420 STOP
```

6. ERROR MESSAGES

At any time during execution of a BASIC statement an error may occur due to something being incorrect or due to some kind of overflow. If an error occurs then you will be informed of the error by an appropriate error message.

Error messages indicate the error type, using an error code, and where the error occurred. They have the following format :

xx ERROR IN aaaa

where xx is the error code, and aaaa is the line number where the error occurred. As previously mentioned for errors which occur in Immediate mode the line number will be displayed as line 0000.

A list of error codes and descriptions is shown in table 5.2. A more complete list including explanations for each error type can be found in Appendix B.

TABLE 5.2 - Error Codes and Descriptions

Error Code	Description
SN	Syntax error
NL	Nonexistent line number
/0	Division by zero
OV	Arithmetic overflow
FN	FOR without NEXT
NF	NEXT without FOR
GR	GOSUB without RETURN
RG	RETURN without GOSUB
OD	Out of data
OM	Out of memory
EE	Error in Expression
SO	Stack overflow
CC	Can't continue

APPENDIX A

DATUM INTEGER BASIC SUMMARY

Cold start address : 2800 hex
Warm start address : 2803 hex

COMMANDS

LIST
NEW
MON
RUN

BASIC STATEMENTS

DATA	NEXT
* END	* POKE
FOR...TO...STEP	* PRINT
GOSUB	READ
* GOTO	* REM
* IF...THEN	* RETURN
INPUT	* STOP
* LET	

(* means statement can be
used in immediate mode.)

FUNCTIONS

PEEK

ARITHMETIC OPERATORS

+
-
*
/
()

LOGICAL OPERATORS

AND
OR
NOT

RELATIONAL OPERATORS

=
<
>

SPECIAL KEYS

<RETURN>	carriage return and line feed
<BREAK>	halt execution / cancel
<CTRL><H>	backspace

FEATURES

Upper case alphabetic characters only
Range of integers : -32768 to +32767
Range of line numbers : 1 to 9999
Variables : A, B, C, ..., Z only (26 in total)
Maximum input line length : 72 characters

MESSAGES

<BASIC>	- Start up message.
!	- Operating system prompt.
?	- INPUT prompt.
STOPPED AT aaaa	- STOP instruction executed or <BREAK> pressed. aaaa = line number of line stopped at.
xx ERROR IN aaaa	- Error type xx found in line aaaa.
BAD INPUT	- Invalid characters entered during INPUT.
WHAT	- Syntax error in operating system command, or command did not make any sense at all.

APPENDIX B

DATUM INTEGER BASIC ERROR CODES

CODE	DESCRIPTION	EXPLANATION
SN	Syntax error	An error in the syntax of a BASIC statement or its operands was found.
NL	Non-existent line number	An attempt was made to GOTO or GOSUB to a line number which does not exist in the current program.
/0	Divide by zero	In a division operation the divisor was zero, implying an undefined result.
OV	Arithmetic overflow	An integer too large to be represented by DATUM BASIC was read in a BASIC line, or produced as a result of calculation.
FN	FOR without NEXT	On reaching the end of a FOR loop a search for the corresponding NEXT statement was unsuccessful.
NF	NEXT without FOR	NEXT was encountered without a matching FOR statement being executed previously.
GR	GOSUB without RETURN	An unused RETURN address was found on the stack during FOR...NEXT operations.
RG	RETURN without GOSUB	A RETURN statement was found without having previously executed a GOSUB statement.
OD	Out of data	A READ was executed without sufficient data to assign to all variables in the READ list.
OM	Out of memory	Your BASIC program has filled all available memory allocated for use. To gain more memory try deleting unnecessary comments in your program.

CODE	DESCRIPTION	EXPLANATION
EE	Error in Expression	Unbalanced brackets in an arithmetic expression.
SO	Stack overflow	Too many open FOR...NEXT loops and/or RETURN addresses and/or open brackets in an arithmetic expression caused the stack to overflow.
CC	Can't continue	RETURN was issued from immediate mode after END was executed or program execution was completed. i.e. no continuable program exists.

APPENDIX C

ASCII CHARACTER SET

CODE		CHARACTER	CODE		CHARACTER
Dec.	Hex.		Dec.	Hex.	
8	08	<backspace>	71	47	G
10	0A	<linefeed>	72	48	H
13	0D	< cr >	73	49	I
32	20	<space>	74	4A	J
33	21	!	75	4B	K
34	22	"	76	4C	L
35	23	#	77	4D	M
36	24	\$	78	4E	N
37	25	%	79	4F	O
38	26	&	80	50	P
39	27	'	81	51	Q
40	28	(82	52	R
41	29)	83	53	S
42	2A	*	84	54	T
43	2B	+	85	55	U
44	2C	,	86	56	V
45	2D	-	87	57	W
46	2E	.	88	58	X
47	2F	/	89	59	Y
48	30	0	90	5A	Z
49	31	1	91	5B	[
50	32	2	92	5C	\
51	33	3	93	5D]
52	34	4	94	5E	^
53	35	5	95	5F	_
54	36	6	96	60	`
55	37	7	97	61	a
56	38	8	98	62	b
57	39	9	99	63	c
58	3A	:	100	64	d
59	3B	;	101	65	e
60	3C	<	102	66	f
61	3D	=	103	67	g
62	3E	>	104	68	h
63	3F	?	105	69	i
64	40	@	106	6A	j
65	41	A	107	6B	k
66	42	B	108	6C	l
67	43	C	109	6D	m
68	44	D	110	6E	n
69	45	E	111	6F	o
70	46	F	112	70	p

CODE		CHARACTER
Dec.	Hex.	
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w

CODE		CHARACTER
Dec.	Hex.	
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	
125	7D	}
126	7E	~

APPENDIX D

SAMPLE RUN

<BASIC>

```
! 10 REM      THIS PROGRAM CONVERTS BASE 10
! 20 REM      INTEGERS TO BINARY.
! 30 REM
! 40 PRINT
! 50 PRINT "ENTER AN INTEGER";
! 60 INPUT D
! 70 IF D < 0 THEN PRINT "INTEGER MUST BE POSITIVE"
! 80 IF D > 16383 THEN PRINT "INTEGER IS TOO LARGE"
! 90 IF D = 0 THEN PRINT "BINARY EQUIVALENT = 0"
! 100 IF (D < 1) OR (D > 16383) THEN GOTO 40
! 110 PRINT "BINARY EQUIVALENT = ";
! 120 LET N = 0
! 130 LET M = 1
! 140 IF M > D THEN GOTO 180
! 150 M = M*2
! 160 N = N + 1
! 170 GOTO 140
! 180 N = N - 2
! 190 D = D - (M/2)
! 200 PRINT "1";
! 210 FOR I = N TO 0 STEP -1
! 220   IF NOT (M > D) THEN D = D - M
! 230   GOSUB 1000
! 240   IF M > D THEN PRINT "0";
! 250   IF NOT (M > D) THEN PRINT "1";
! 260 NEXT I
! 270 PRINT
! 280 GOTO 40
! 290 END
! 1000 REM     THIS SUBROUTINE CALCULATES 2 TO THE
! 1010 REM     POWER I AND RETURNS THE RESULT IN M
! 1020 REM
! 1030 LET M = 1
! 1040 FOR J = 1 TO I
! 1050   M = M*2
! 1060 NEXT J
! 1070 RETURN
! LIST

0010 REM      THIS PROGRAM CONVERTS BASE 10
0020 REM      INTEGERS TO BINARY.
0030 REM
0040 PRINT
0050 PRINT "ENTER AN INTEGER";
0060 INPUT D
0070 IF D < 0 THEN PRINT "INTEGER MUST BE POSITIVE"
0080 IF D > 16383 THEN PRINT "INTEGER IS TOO LARGE"
0090 IF D = 0 THEN PRINT "BINARY EQUIVALENT = 0"
0100 IF (D < 1) OR (D > 16383) THEN GOTO 40
0110 PRINT "BINARY EQUIVALENT = ";
```

```

0120 LET N = 0
0130 LET M = 1
0140 IF M > D THEN GOTO 180
0150 M = M*2
0160 N = N + 1
0170 GOTO 140
0180 N = N - 2
0190 D = D - (M/2)
0200 PRINT "1";
0210 FOR I = N TO 0 STEP -1
0220 IF NOT (M > D) THEN D = D - M
0230 GOSUB 1000
0240 IF M > D THEN PRINT "0";
0250 IF NOT (M > D) THEN PRINT "1";
0260 NEXT I
0270 PRINT
0280 GOTO 40
0290 END
1000 REM      THIS SUBROUTINE CALCULATES 2 TO THE
1010 REM      POWER I AND RETURNS THE RESULT IN M
1020 REM
1030 LET M = 1
1040 FOR J = 1 TO I
1050 M = M*2
1060 NEXT J
1070 RETURN

```

! RUN

ENTER AN INTEGER? -20
 INTEGER MUST BE POSITIVE

ENTER AN INTEGER? 16384
 INTEGER IS TOO LARGE

ENTER AN INTEGER? 0
 BINARY EQUIVALENT = 0

ENTER AN INTEGER? 1
 BINARY EQUIVALENT = 1

ENTER AN INTEGER? 2
 BINARY EQUIVALENT = 10

ENTER AN INTEGER? 10
 BINARY EQUIVALENT = 1010

ENTER AN INTEGER? 123
 BINARY EQUIVALENT = 1111011

ENTER AN INTEGER? 15000
 BINARY EQUIVALENT = 11101010011000

ENTER AN INTEGER? <BREAK>

STOPPED AT 0060

APPENDIX E

GLOSSARY

Address : A location in memory, specified by a number (in hexadecimal usually).

Alphabetic : Strictly the letters of the alphabet - A,B,...,Z.

ASCII : American Standard Code for Information Interchange.
Refers basically to a character set with each character having its own unique, standard code.

Assignment : The allocation of a value to a variable i.e. assign a value to a variable.

BASIC : Beginners All-purpose Symbolic Instruction Code. A programming language developed to allow computers to be programmed using English-like instructions.

Backspace : Movement of the cursor back one character usually in order to retype that character.

Binary : A representation of data using only 2 states e.g. : the digits 0 and 1. This ultimately is the representation a computer uses internally with the two states being represented by a high and a low voltage.

Byte : One unit of memory capable of representing 256 different values. One way in which a byte is often defined is to say that 1 byte can store 1 character.

Constant : A value which is not subject to change.

Carriage return : A character which returns the cursor to the left hand side of the screen on the current line.

Command : An instruction entered by the user which tells the computer to do something.

Cursor : An indicator on the display showing where the next character will go when typed at the keyboard.

Data : Information transferred to and from a program.

Decrement : To set a value down by an amount (usually by 1).

Delimiter : A character which marks the beginning or end of a piece of information but is not part of that information.

EPROM : Erasable Programmable Read Only Memory. Memory which can have programs burnt into it and can be erased again by exposing the memory to ultraviolet light.

Expression : An arrangement of constants, variables and functions all linked by operators.

Function : A subroutine which performs some computation on data and returns a result to the place of call.

Hardware : The physical parts of a computer system such as the processor, keyboard, display, printer, disk drives, etc.

Hex : See hexadecimal

Hexadecimal : A representation for a number system using base 16. Numbers in hexadecimal are usually written using the digits : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Hexpad : A keyboard or keypad which contains only the digits for entering hexadecimal values.

Immediate mode : A mode in an interpreter in which all input is processed immediately on entry rather than being stored for later processing.

Increment : To set a value up by an amount (usually by 1).

Infinite loop : A loop which continues over and over again indefinitely.

Input : The transfer of data from the outside world into the computer system (mostly via a keyboard).

Integer : A number which is whole, i.e. has no fractional part.

Interpreter : A computer program that translates, then executes another computer program, one step at a time.

Kbyte : see kilobyte.

Kilobyte : 1024 bytes of memory.

Linefeed : A character which advances the cursor one line down the screen (or scrolls the entire screen contents up one line if already at the bottom of the screen).

Line number : A number which references a particular line in a BASIC program and acts as an identifier or addressing value for that line. Line numbers also maintain the sequence of a BASIC program.

Loop : A section of a program which is executed over and over again.

Lower case : See upper case.

Machine code : The instruction set of a machine usually represented in hexadecimal. The machine understandable language which all other languages are translated into prior to execution.

Monitor : The program which is run automatically on power up of the computer and which controls the users operations with the computer.

Operands : Values which go with an instruction in order to define the parameters for the operations of that instruction.

Operating system : The program which controls, directs and implements the user's operations with the computer and makes these operations easily specified from the user's point of view.

Operator : A symbol or function which takes one or more operands and performs some action on them.

Output : The transfer of data from within the computer back to the user (mostly by display screen or printer).

Program : An ordered set of instructions which tell the computer what to do, when to do it, and how to do it.

Prompt : A message or symbol provided by the program to indicate that it is ready to accept input.

RAM : Random Access Memory. Memory which can have information read from or written to it.

ROM : Read Only Memory. Memory which contains permanent information which can be read but cannot be written to or altered in any way.

Stack : A special section of memory set up to act as a last in - first out queue. The stack is used mainly for storing addresses and arithmetic values for later use.

Statement : A complete instruction in BASIC.

Subroutine : A section of a program which is made common to all other parts of the program and which is called by a single instruction, returning back to the place of call after completion of its operations.

Syntax : The "grammatical" construct of a command or statement.

Unary operator : An operator which acts on one value only.

Upper case : The set of capital alphabetic characters as opposed to lower case being the set of "small letter" alphabetic characters.

Variable : A place in memory where data is stored, often thought of conceptually as a storage box.

ASSEMBLY LISTING OF INTERPRETER SOURCE CODE

ASSEMBLY LISTING OF INTERPRETER SOURCE CODE

MOTOROLA M68SAM CROSS-ASSMBLER

PAGE 1

M6800 CROSS ASSEMBLER , R.M.I.T.

```

00001          NAM      BASINT
00002          *****
00003          *
00004          *
00005          *          B A S I C      I N T E R P R E T E R
00006          *          -----
00007          *
00008          *          WRITTEN BY SCOTT GOGLER
00009          *
00010          *
00011          * PROGRAM FUNCTION :
00012          *
00013          *          THIS PROGRAM IMPLEMENTS THE MAJOR FUNCTIONS
00014          * OF AN INTEGER BASIC LANGUAGE INTERPRETER
00015          * SPECIFICALLY FOR THE "DATUM" MICROCOMPUTER KIT.
00016          *
00017          *****
00018          0008      BACKSP EQU      $08      BACKSPACE CHARACTER
00019          000D      CR      EQU      $0D      CARRIAGE RETURN CHARACTER
00020          0010      ENDADD EQU      $0010      END ADDR (USED BY MONITOR)
00021          0004      EOT      EQU      $04      END-OF-TEXT CHARACTER
00022          722F      FILMEM EQU      $722F      UTILITY ADDR FOR FILLING MEM
00023          70DA      IN      EQU      $70DA      UTILITY ADDR FOR INPUT A CHAR
00024          000A      LF      EQU      $0A      LINE FEED CHARACTER
00025          000F      LSMASK EQU      $0F      MASK TO RETAIN LSPART OF BYTE
00026          2000      MEMLIM EQU      $2000      ADDR OF MAX LIMIT USER PROG
00027          7252      MOVMEM EQU      $7252      UTILITY ADDR FOR MOVING MEM
00028          00F0      MSMASK EQU      $F0      MASK TO RETAIN MSPART OF BYTE
00029          0080      NEGMSK EQU      $80      MASK TO DETERMINE IF NEG NO.
00030          0000      NMIV      EQU      $0000      ADDR OF NMI INTERRUPT VECTOR
00031          73CB      NXTSPC EQU      $73CB      UTILITY ADDR FOR OUTPUT SPCS
00032          70E7      OUT      EQU      $70E7      UTILITY ADDR FOR OUTPUT CHAR
00033          70C5      OUTS      EQU      $70C5      UTILITY ADDR FOR PRINT SPACES
00034          7402      PCRLF      EQU      $7402      UTILITY ADDR FOR PRINT CR, LF
00035          1001      PROGST EQU      $1001      START ADDR OF USERS PROG
00036          703F      PSTRNG EQU      $703F      UTILITY ADDR FOR PRINT STRING
00037          7008      RESET      EQU      $7008      DATERM MONITOR RESET ADDR
00038          000E      STARAD EQU      $000E      START ADDR (USED BY MONITOR)
00039          2700      STKLIM EQU      $2700      MAX LIMIT OF USERS STACK TOP
00040          0008      TABSIZ EQU      $08      SIZE OF TAB PRINT ZONE

```

00042	27FF	USRSTK EQU	\$27FF	ADDR OF BOTTOM OF USERS STACK
00043	2000	VARBEG EQU	\$2000	BEGIN ADDR OF USER VAR AREA
00044	2033	VAREND EQU	\$2033	END ADDR OF USER VAR AREA
00045	0006	XTEMP EQU	\$0006	DESTIN ADDR (USED BY MONITOR)
00046		*		
00047		*	ERROR MESSAGE CODES	
00048		*	-----	
00049		*		
00050	534E	SYNTAX EQU	\$534E	(SN) SYNTAX ERROR
00051	4E4C	NOLINE EQU	\$4E4C	(NL) NONEXISTENT LINE NO.
00052	2F30	DIVZER EQU	\$2F30	(/0) DIVISION BY ZERO
00053	4F56	ARITOV EQU	\$4F56	(OV) ARITHMETIC OVERFLOW
00054	464E	FORNEX EQU	\$464E	(FN) FOR WITHOUT NEXT
00055	4E46	NEXFOR EQU	\$4E46	(NF) NEXT WITHOUT FOR
00056	4752	GOSRET EQU	\$4752	(GR) GOSUB WITHOUT RETURN
00057	5247	RETGOS EQU	\$5247	(RG) RETURN WITHOUT GOSUB
00058	4F44	OUTDAT EQU	\$4F44	(OD) OUT OF DATA
00059	4F4D	OUTMEM EQU	\$4F4D	(OM) OUT OF MEMORY
00060	4545	EXPERR EQU	\$4545	(EE) ERROR IN EXPRESSION
00061	534F	STAKOV EQU	\$534F	(SO) STACK OVERFLOW
00062	4343	NOCNT EQU	\$4343	(CC) CAN'T CONTINUE
00063		*		
00064		*	INTERPRETER WORK AREA	
00065		*	-----	
00066		*		
00067	2034	ORG	\$2034	
00068	2034 0048	BUFFER RMB	72	INPUT BUFFER
00069		*		
00070	207C	ORG	\$207C	
00071	207C 0001	BRKFLG RMB	1	<BREAK> FLAG
00072	207D 0002	BUFPTR RMB	2	POINTER TO INPUT BUFFER
00073	207F 0001	COUNT RMB	1	GENERAL PURPOSE COUNTING BYTE
00074	2080 0002	DATPTR RMB	2	PTR TO DATA ITEM LAST READ
00075	2082 0002	ENDVAL RMB	2	END VALUE FOR LOOPS
00076	2084 0003	ERRCOD RMB	3	ERROR CODE + EOT CHARACTER
00077	2087 0005	ERRLIN RMB	5	ASCII ERROR LINE NO. + EOT
00078	208C 0001	FLAG RMB	1	GENERAL PURPOSE FLAG
00079	208D 0002	FNDPTR RMB	2	PTR TO A LINE FOUND IN PROG
00080	208F 0001	LASTCH RMB	1	LAST CHAR PROCESSED IN PRINT
00081	2090 0001	LENGTH RMB	1	STORAGE FOR LENGTH OF A LINE
00082	2091 0002	LINENO RMB	2	STORAGE FOR BCD LINE NO.
00083	2093 0002	LRGLIN RMB	2	LARGEST LINE NO. IN PROG
00084	2095 0001	LSPART RMB	1	LS PART OF BCD LINE NO.
00085	2096 0001	MSPART RMB	1	MS PART OF BCD LINE NO.
00086	2097 0005	NUMBER RMB	5	ASCII REPRESENTATION OF A NO.
00087	209C 0002	OPRND1 RMB	2	OPERAND USED IN ARITHMETIC
00088	209E 0002	PRGCNT RMB	2	BASIC LINE NO. PROG COUNTER
00089	20A0 0002	PROGEN RMB	2	END ADDR OF USER PROG
00090	20A2 0002	RESULT RMB	2	RESULT OF EVALUATED EXPRSN
00091	20A4 0002	RUNPTR RMB	2	ADDR NEXT BASIC LINE TO RUN
00092	20A6 0001	SIGN RMB	1	SIGN OF AN ASCII NUMBER
00093	20A7 0002	SP RMB	2	SYSTEM STACK PTR TEMP STORAGE

00094	20A9	0001	TABPOS	RMB	1	TAB POSITION FOR PRINTING
00095	20AA	0002	TBLPTR	RMB	2	PTR TO LOOK UP TABLES
00096	20AC	0002	TEMP	RMB	2	TEMPORARY INTEGER STORAGE
00097	20AE	0001	UNOPTR	RMB	1	UNARY OPERATOR FLAG
00098	20AF	0002	USRSP	RMB	2	USER'S STACK POINTER
00099	20B1	0001	VAR	RMB	1	STORAGE FOR A VARIABLE NAME
00100	20B2	0001	VARNAM	RMB	1	STORAGE FOR A VARIABLE NAME
00101	20B3	0002	XTEMP0	RMB	2	INDEX REG TEMP STORAGE
00102	20B5	0002	XTEMP1	RMB	2	INDEX REG TEMP STORAGE
00103	20B7	0002	XTEMP2	RMB	2	INDEX REG TEMP STORAGE
00104	20B9	0002	XTEMP3	RMB	2	INDEX REG TEMP STORAGE
00105	20BB	0002	XTEMP4	RMB	2	INDEX REG TEMP STORAGE
00106	20BD	0002	XTEMP5	RMB	2	INDEX REG TEMP STORAGE

00107

*

00108

*

00109

*

BASIC INTERPRETER MAINLINE

00110

*

00111

*

00112

*

THIS ROUTINE PERFORMS EITHER A WARM OR COLD

00113

*

START, DEPENDING ON WHICH ENTRY POINT EXECUTION

00114

*

BEGINS AT, THEN BEGINS THE INTERPRETER PROCESS.

00115

*

00116

00117 2800

ORG

\$2800

00118 2800 BD 280F

START

JSR

INTLZE

PERFORM INITIALIZATION

00119 2803 CE 37B4

RESTR

LDX

#STRUP

DISPLAY START

00120 2806 BD 703F

JSR

PSTRNG

UP MESSAGE

00121 2809 BF 20A7

STS

SP

SAVE STACK POINTER

00122 280C 7E 284E

JMP

INTPRT

BEGIN INTERPRETER PROCESS

00123

*

00124

*

00125

*

INITIALIZATION

00126

*

00127

*

00128

*

THIS ROUTINE PERFORMS ALL THE NECESSARY

00129

*

INITIALIZATION FOR A COLD START. IT ALSO DOUBLES

00130

*

AS THE PROCESS FOR THE "NEW" COMMAND.

00131

*

00132

00133 280F 0F

INTLZE

SEI

DISABLE <BREAK> KEY

00134 2810 CE 37AE

LDX

#BRKISR

SET UP INTERRUPT

00135 2813 DF 00

STX

NMIV

VECTOR

00136 2815 86 04

LDA

#EOT

SET UP END OF TEXT CHAR

00137 2817 B7 2086

STA

ERRCOD+2

AT END OF ERROR CODE AND

00138 281A B7 208B

STA

ERRLIN+4

AT END OF ERROR LINE NO.

00139 281D 20 06

BRA

NEW1

SKIP KEYWORD CHECK FOR "NEW"

00140 281F FE 20B5

NEW

LDX

XTEMP1

RETRIEVE SAVED BUFFER PTR

00141 2822 BD 3799

JSR

CHKCR1

CHECK FOR <CR> AFTER KEYWORD

00142 2825 CE 27FF

NEW1

LDX

#USRSTK

SET UP USERS

00143 2828 FF 20AF

STX

USRSP

STACK POINTER

00144 282B CE 1001

LDX

#PROGST

CLEAR USER

00145 282E FF 20A0

STX

PROGEN

PROGRAM AREA

```

00146 2831 CE 2000      LDX      #VARBEG      SET ALL
00147 2834 DF 0E        STX      STARAD
00148 2836 CE 2033      LDX      #VAREND      USER VARIABLES
00149 2839 DF 10        STX      ENDADD
00150 283B 4F           CLR      A              TO ZERO
00151 283C BD 722F      JSR      FILMEM
00152 283F 7F 207C      CLR      BRKFLG      NO <BREAK> OCCURRED
00153 2842 7F 2093      CLR      LRGLIN      SET LARGEST LINE
00154 2845 7F 2094      CLR      LRGLIN+1    NUMBER = 0000
00155 2848 86 0D        LDA      A      #CR      SET UP BYTE IMMEDIATELY
00156 284A B7 1000      STA      A      PROGST-1    BEFORE USER PROG TO BE <CR>
00157 284D 39          RTS
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170 284E 7F 2091      INTPRT CLR      LINENO      SET LINE NUMBER = 0000
00171 2851 7F 2092      CLR      LINENO+1
00172 2854 86 01        LDA      A      #$01      SET TAB POSITION
00173 2856 B7 20A9      STA      A      TABPOS      BACK TO 1 AGAIN
00174 2859 CE 37C2      LDX      #MESG1      DISPLAY MAIN PROMPT
00175 285C BD 703F      JSR      PSTRNG
00176 285F 8D 18        BSR      FILBUF      LOAD BUFFER FROM KEYBOARD
00177 2861 CE 2034      LDX      #BUFFER
00178 2864 BD 377A      JSR      SKPSPC      SKIP ANY SPACES
00179 2867 BD 3746      JSR      CLASS      CLASSIFY FIRST NONSPACE CHAR
00180 286A C1 01        CMP      B      #$01      IF CHAR IS NON NUMERIC
00181 286C 27 07        BEQ      EDMODE
00182 286E BD 7402      JSR      PCRLF      ADVANCE ONE LINE
00183 2871 8D 34        BSR      OPSYS      THEN BEGIN OPERATING SYS MODE
00184 2873 20 DA        BRA      INTPRT
00185 2875 8D 5E        EDMODE BSR      EDITOR      OTHERWISE BEGIN EDITOR MODE
00186 2877 20 D6        BRA      INTPRT      REPEAT INTERPRETER PROCESS
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197 2879 0E          FILBUF CLI      ENABLE <BREAK> KEY

```

```

00198 287A CE 2034      LDX      #BUFFER    SET PTR AT START OF BUFFER
00199 287D BD 70DA FILBF1 JSR      IN        READ ONE CHARACTER
00200 2880 81 08        CMP A    #BACKSP    IF CHAR IS BACKSPACE AND
00201 2882 26 08        BNE      FILBF2
00202 2884 8C 2034      CPX      #BUFFER    NOT AT START OF BUFFER
00203 2887 27 F5        BEQ      FILBF1
00204 2889 09          DEX
00205 288A 20 F2        BRA      FILBF1    THEN DECREMENT BUFFER POINTER
00206 288C 7D 207C FILBF2 TST      BRKFLG   AND GET NEXT CHARACTER
00207 288F 27 03        BEQ      FILBF3    IF <BREAK> PRESSED THEN
00208 2891 7E 3156      JMP      BREAK   JUMP TO
00209 2894 A7 00      FILBF3 STA A    0,X    BREAK ROUTINE
00210 2896 0B          INX              STORE CHAR IN BUFFER AND
00211 2897 81 0D        CMP A    #CR      GET NEXT BUFFER POSITION
00212 2899 27 0A        BEQ      FILBF4    IF CHAR IS A <CR>
00213 289B 8C 207C      CPX      #BUFFER+72 THEN BUFFER FILLED
00214 289E 26 DE        BNE      FILBF1    IF BUFFER SIZE NOT EXCEED
00215 28A0 09          DEX              THEN READ NEXT CHAR
00216 28A1 86 0D        LDA A    #CR      OTHERWISE CHANGE LAST
00217 28A3 A7 00      STA A    0,X    CHARACTER TO
00218 28A5 0F      FILBF4 SEI              A <CR>
00219 28A6 39          RTS              DISABLE <BREAK> KEY
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234 2BA7 FF 207D OPSYS STX      BUFPTR   STORE BUFFER POINTER
00235 28AA CE 2A3A      LDX      #COMTBL   SET UP TABLE POINTER AT
00236 28AD FF 20AA      STX      TBLPTR    START COMMAND KEYWORD TABLE
00237 28B0 FE 207D      LDX      BUFPTR    RETRIEVE BUFFER POINTER
00238 28B3 BD 2A00      JSR      FINDKE   ATTEMPT TO FIND COMMAND
00239 28B6 5D          TST B              IF COMMAND NOT FOUND THEN
00240 28B7 26 03        BNE      OPSYS1
00241 28B9 8D 04        BSR      IMMEDI    EXECUTE IMMEDIATE MODE
00242 28BB 39          RTS
00243 28BC AD 00 OPSYS1 JSR      0,X      OTHERWISE EXECUTE COMMAND
00244 28BE 39          RTS
00245
00246
00247
00248
00249

```

```

00250      *      THIS ROUTINE LOOKS TO SEE IF THE USER INPUT IS
00251      *      A VALID IMMEDIATE MODE KEYWORD AND EXECUTES IT IF
00252      *      IT IS, OTHERWISE THE INPUT IS ASSUMED TO BE AN
00253      *      EXPRESSION AND SO CONTROL IS TRANSFERRED TO THE
00254      *      EXPRESSION HANDLER.
00255      *
00256      *      I/P :
00257      *      BUFPTR = POINTER TO CURRENT BUFFER POSITION
00258      *
00259      *****
00260 28BF CE 2A6D IMMEDI LDX      #IMMTBL    SET UP TABLE POINTER AT
00261 28C2 FF 20AA      STX      TBLPTR    START IMMEDIATE KEYWORD TABLE
00262 28C5 FE 207D      LDX      BUFPTR    RETRIEVE BUFFER POINTER
00263 28C8 BD 2A00      JSR      FINDKE    ATTEMPT TO FIND KEYWORD
00264 28CB 5D          TST B          IF KEYWORD NOT FOUND THEN
00265 28CC 26 04          BNE      IMMEDI
00266 28CE BD 2E7E      JSR      EXPRSN    PROCESS EXPRESSION
00267 28D1 39          RTS
00268 28D2 AD 00 IMMEDI JSR      0,X      OTHERWISE EXECUTE KEYWORD
00269 28D4 39          RTS
00270      *
00271      *
00272      *      EDITOR
00273      *      -----
00274      *
00275      *      THIS ROUTINE USES THE USER INPUT TO ALTER THE
00276      *      CURRENT BASIC PROGRAM IN MEMORY ACCORDINGLY. IT
00277      *      EITHER AMENDS AN OLD LINE, INSERTS A NEW LINE,
00278      *      DELETES AN OLD LINE OR APPENDS A NEW LINE TO THE
00279      *      END OF THE CURRENT USER PROGRAM.
00280      *
00281      *      I/P :
00282      *      IX = POINTER TO FIRST NONSPACE CHAR IN BUFFER
00283      *
00284      *****
00285 28D5 BD 35E1 EDITOR JSR      NUMBCD    READ LINE NO., CONVERT TO BCD
00286 28D8 7D 2096      TST      MSPART    IF LINE NUMBER IS
00287 28DB 26 08          BNE      LINEOK
00288 28DD 7D 2095      TST      LSPART    0000 THEN DISPLAY
00289 28E0 26 03          BNE      LINEOK
00290 28E2 7E 31C5      JMP      WHAT      "WHAT?" ERROR MESSAGE
00291 28E5 BD 377A LINEOK JSR      SKPSPC    SKIP TO FIRST NONSPACE CHAR
00292 28E8 FF 207D      STX      BUFPTR    STORE BUFFER POINTER
00293 28EB 86 0D          LDA A      #CR      IF FIRST CHAR IS A <CR>
00294 28ED A1 00          CMP A      0,X
00295 28EF 27 35          BEQ      EDITR5    THEN JUMP TO DELETE LINE
00296 28F1 B6 2096      LDA A      MSPART    IF LINE NO. IS DIFFERENT
00297 28F4 B1 2093      CMP A      LRGLIN    THAN LARGEST LINE NO.
00298 28F7 22 2A          BHI      EDITR4
00299 28F9 26 08          BNE      EDITR1    THEN JUMPTO INSERT/AMEND LINE
00300 28FB B6 2095      LDA A      LSPART
00301 28FE B1 2094      CMP A      LRGLIN+1

```



```

00302 2901 22 20      BHI     EDITR4  OTHERWISE JUMP TO APPEND LINE
00303 2903 BD 3580 EDITR1 JSR     FINDLN  ATTEMPT TO FIND LINE IN PROG
00304 2906 5D          TST     B        IF LINE ALREADY EXISTS THEN
00305 2907 26 02      BNE     EDITR2
00306 2909 8D 60      BSR     DELET1  DELETE OLD LINE
00307 290B FE 207D EDITR2 LDX     BUFPTR
00308 290E B6 2096      LDA     MSPART  TEST TO SEE IF THE
00309 2911 B1 2093      CMP     A     LRGLIN LINE DELETED WAS THE
00310 2914 25 0A      BCS     EDITR3  LAST LINE OF THE PROG
00311 2916 B6 2095      LDA     A     LSPART AND IF SO MUST NOW
00312 2919 B1 2094      CMP     A     LRGLIN+1 APPEND NEW LINE
00313 291C 22 05      BHI     EDITR4  OTHERWISE
00314 291E 27 03      BEQ     EDITR4
00315 2920 8D 07      BSR     EDITR3  INSERT NEW LINE
00316 2922 39          RTS          AND EXIT
00317 2923 8D 29      EDITR4 BSR     APPEND  APPEND LINE
00318 2925 39          RTS          AND EXIT
00319 2926 8D 3A      EDITR5 BSR     DELETE  DELETE LINE
00320 2928 39          RTS          AND EXIT
00321          *
00322          *
00323          *      INSERT LINE
00324          *      -----
00325          *
00326          *      THIS ROUTINE INSERTS A NEW LINE INTO THE
00327          *      CORRECT POSITION WITHIN THE USER'S CURRENT PROGRAM
00328          *      ,AND UPDATES PROGRAM PARAMETERS ACCORDINGLY.
00329          *
00330          *      I/P :
00331          *      IX = POINTER TO START BASIC STATEMENT IN BUFFER
00332          *      BUFPTR = SAME AS IX EXCEPT STORED
00333          *      FNDPTR = PTR TO POSITION FOR INSERTION OF LINE
00334          *      MSPART = MS BYTE OF BCD LINE NUMBER
00335          *      LSPART = LS BYTE OF BCD LINE NUMBER
00336          *
00337          *      *****
00338 2929 FF 20B3 INSERT STX     XTEMP0  STORE POINTER
00339 292C FE 20A0      LDX     PROGEN  SET UP END ADDR OF PROG
00340 292F DF 10      STX     ENDAID  BLOCK TO BE MOVED
00341 2931 FE 20B3      LDX     XTEMP0  RETRIEVE POINTER
00342 2934 BD 29B1      JSR     UPDEND  UPDATE PROGRAM END POINTER
00343 2937 FE 208D      LDX     FNDPTR  SET UP START ADDR OF PROG
00344 293A DF 0E      STX     STARAD  BLOCK TO BE MOVED
00345 293C F6 2090      LDA     B     LENGTH CALCULATE DESTINATION ADDR
00346 293F BD 375D      JSR     IXADD  OF BLOCK TO BE MOVED
00347 2942 DF 06      STX     XTEMP
00348 2944 BD 7252      JSR     MOVMEM  MOVE PROG BLOCK TO CREATE GAP
00349 2947 FE 208D      LDX     FNDPTR  GET PTR TO START OF NEW LINE
00350 294A BD 29D7      JSR     ADDLN  ADD LINE TO GAP IN USER PROG
00351 294D 39          RTS
00352          *
00353          *

```

```

00354          *          APPEND LINE
00355          *          -----
00356          *
00357          *          THIS ROUTINE APPENDS A NEW LINE TO THE END OF
00358          *          THE USER'S CURRENT PROGRAM, AND UPDATES PROGRAM
00359          *          PARAMETERS ACCORDINGLY.
00360          *
00361          *          I/P :
00362          *          IX = POINTER TO START BASIC STATEMENT IN BUFFER
00363          *          BUFPTR = SAME AS IX EXCEPT STORED
00364          *          MSPART = MS BYTE OF BCD LINE NUMBER
00365          *          LSPART = LS BYTE OF BCD LINE NUMBER
00366          *
00367          *****
00368 294E 8D 61  APPEND BSR      UPDEND      UPDATE PROG END POINTER
00369 2950 B6 2096      LDA A      MSPART      UPDATE LARGEST LINE NUMBER
00370          *
00371 2953 B7 2093      STA A      LRGLIN
00372          *
00373 2956 B6 2095      LDA A      LSPART      TO CURRENT LINE NUMBER
00374          *
00375 2959 B7 2094      STA A      LRGLIN+1
00376          *
00377 295C FE 20B3      LDX      XTEMP0      GET PTR TO START OF NEW LINE
00378          *
00379 295F 8D 76      SSR      ADDLN      ADD NEW LINE TO USER PROGRAM
00380          *
00381 2961 39          RTS
00382          *
00383          *
00384          *          DELETE LINE
00385          *          -----
00386          *
00387          *          THIS ROUTINE DELETES AN OLD LINE FROM THE USER
00388          *          PROGRAM GIVEN THE LINE NUMBER, AND UPDATES PROGRAM
00389          *          PARAMETERS (SUCH AS PROG END AND LARGEST LINE NO.)
00390          *          ACCORDINGLY.
00391          *
00392          *          I/P :
00393          *          MSPART = MS BYTE OF BCD LINE NUMBER
00394          *          LSPART = LS BYTE OF BCD LINE NUMBER
00395          *
00396          *****
00397 2962 BD 3580  DELETE JSR      FINDLN      ATTEMPT TO FIND LINE IN PROG
00398          *
00399 2965 5D          TST B
00400          *
00401 2966 27 03      BEQ      DELET1      IF LINE NOT FOUND THEN
00402          *
00403 2968 7E 31C5      JMP      WHAT      DISPLAY "WHAT?" ERROR MESSAGE
00404          *
00405 296B DF 06      DELET1 STX      XTEMP      STORE DESTINATION ADDRESS
00406          *
00407 296D B6 2093      LDA A      LRGLIN      TEST IF LINE TO BE
00408          *
00409 2970 A1 00      CMP A      0,X      DELETED IS THE
00410          *
00411 2972 26 1F      BNE      DELET4      LARGEST LINE
00412          *
00413 2974 B6 2094      LDA A      LRGLIN+1      IF NOT THEN
00414          *
00415 2977 A1 01      CMP A      1,X      CONTINUE AS NORMAL
00416          *
00417 2979 26 18      BNE      DELET4      OTHERWISE :
00418          *
00419 297B 09          DEX
00420          *
00421 297C 86 0D      LDA A      #CR
00422          *
00423 297E 09      DELET2 DEX      DETERMINE NEXT
00424          *
00425 297F A1 00      CMP A      0,X

```

```

00406 2981 26 FC          BNE      DELET2   LARGEST LINE NO.
00407 2983 09          DEX
00408 2984 09          DEX              AND UPDATE
00409 2985 09          DEX
00410 2986 A1 00        CMP A  0,X        THE LARGEST
00411 2988 27 03        BEQ      DELET3
00412 298A 08          INX              LINE NUMBER
00413 298B 08          INX
00414 298C 08          INX              TO BE THIS
00415 298D 08          DELET3 INX
00416 298E EE 00        LDX      0,X
00417 2990 FF 2093      STX      LRGLIN
00418 2993 DE 06        DELET4 LDX      XTEMP   RETRIEVE DESTINATION PTR
00419 2995 E6 02        LDA B  2,X        FIND START ADDRESS
00420 2997 F7 2090      STA B  LENGTH    OF BLOCK TO
00421 299A BD 375D      JSR      IXADD     BE MOVED
00422 299D DF 0E        STX      STARAD   AND STORE
00423 299F FE 20A0      LDX      PROGEN    FIND END ADDRESS
00424 29A2 DF 10        STX      ENDADD   AND STORE
00425 29A4 F6 2090      LDA B  LENGTH    UPDATE END ADDRESS
00426 29A7 BD 3765      JSR      IXSUB
00427 29AA FF 20A0      STX      PROGEN    OF USER'S PROGRAM
00428 29AD BD 7252      JSR      MOVMEM    MOVE REST OF PROG OVER LINE
00429 29B0 39          RTS

```

```

00430          *
00431          *
00432          *      UPDATE END OF PROGRAM PARAMETER
00433          *      -----
00434          *
00435          *      THIS ROUTINE CALCULATES AND STORES THE LENGTH
00436          *      OF THE PROGRAM LINE AND USES THIS TO UPDATE THE
00437          *      END OF PROGRAM POINTER.
00438          *
00439          *      I/P :
00440          *      IX = POINTER TO START BASIC STATEMENT IN BUFFER
00441          *
00442          *      Q/P :
00443          *      XTEMP0 = OLD END OF PROGRAM POINTER
00444          *      LENGTH = LENGTH OF BASIC LINE
00445          *
00446          *      *****
00447 29B1 C6 04          UPDEN0 LDA B  #4      CALCULATE LENGTH
00448 29B3 86 0D          LDA A  #CR
00449 29B5 5C          UPDEN1 INC B
00450 29B6 08          INX              OF BASIC LINE
00451 29B7 A1 00        CMP A  0,X
00452 29B9 26 FB        BNE      UPDEN1
00453 29BB F7 2090      STA B  LENGTH    AND STORE LENGTH
00454 29BE FE 20A0      LDX      PROGEN
00455 29C1 FF 20B3      STX      XTEMP0   SAVE OLD PROG END POINTER
00456 29C4 08          UPDEN2 INX      INCREMENT PROG END POINTER
00457 29C5 8C 2001      CPX      #MEMLIN+1 IF PTR EXCEEDS LIMIT THEN

```

```

00458 29C8 26 06      BNE      UPDEN3
00459 29CA CE 4F4D     LDX      #OUTMEM    SET UP ERROR CODE
00460 29CD 7E 31A1     JMP      ERROR      AND JUMP TO ERROR ROUTINE
00461 29D0 5A          UPDEN3 DEC B    IF NOT FINISHED THEN
00462 29D1 26 F2      BNE      UPDEN2     GO AND INCREMENT PTR AGAIN
00463 29D3 FF 20A0     STX      PROGEN    OTHERWISE STORE NEW PROG END
00464 29D6 39          RTS              AND EXIT
00465                *
00466                *
00467                *      ADD LINE TO PROGRAM
00468                *      -----
00469                *
00470                *      THIS ROUTINE WRITES THE NEW PROGRAM LINE INTO
00471                *      THE SPECIFIED POSITION IN THE CURRENT USER
00472                *      PROGRAM.
00473                *
00474                *      I/P :
00475                *      IX = PTR TO POSITION OF INSERTION OF NEW LINE
00476                *      BUFPTR = POINTER TO BUFFER POSITION
00477                *      MSPART = MS PART OF BCD LINE NUMBER
00478                *      LSPART = LS PART OF BCD LINE NUMBER
00479                *      LENGTH = LENGTH OF BASIC LINE
00480                *
00481                *      *****
00482 29D7 B6 2096 ADDLN LDA A MSPART  WRITE MSBYTE OF LINE
00483 29DA A7 00      STA A 0,X      NUMBER TO PROGRAM
00484 29DC 08          INX              INCREMENT PROG POINTER
00485 29DD B6 2095     LDA A LSPART    WRITE LS BYTE OF LINE
00486 29E0 A7 00      STA A 0,X      NUMBER TO PROGRAM
00487 29E2 08          INX              INCREMENT PROG POINTER
00488 29E3 B6 2090     LDA A LENGTH   WRITE LENGTH OF NEW LINE
00489 29E6 A7 00      STA A 0,X      TO PROGRAM
00490 29E8 08          INX              INCREMENT PROG POINTER
00491 29E9 FF 20B3 ADDLN1 STX XTEMP0    STORE PROG POINTER
00492 29EC FE 207D     LDX BUFPTR      RETRIEVE BUFFER POINTER
00493 29EF A6 00      LDA A 0,X      GET CHARACTER FROM BUFFER
00494 29F1 08          INX              MOVE BUFFER PTR TO NEXT CHAR
00495 29F2 FF 207D     STX BUFPTR      STORE BUFFER POINTER
00496 29F5 FE 20B3     LDX XTEMP0      RETRIEVE PROG POINTER
00497 29F8 A7 00      STA A 0,X      WRITE CHAR TO PROGRAM
00498 29FA 08          INX              MOVE PROG PTR TO NEXT POSN
00499 29FB 81 0D      CMP A #CR        IF NOT END LINE TO BE WRITTEN
00500 29FD 26 EB      BNE ADDLN1      THEN GO AND WRITE NEXT CHAR
00501 29FF 39          RTS              OTHERWISE EXIT
00502                *
00503                *
00504                *      FIND KEYWORD
00505                *      -----
00506                *
00507                *      THIS ROUTINE ATTEMPTS TO FIND A REQUIRED
00508                *      KEYWORD IN THE LOOKUP TABLES, GIVEN THE TABLE IN
00509                *      WHICH TO SEARCH FOR THE KEYWORD.

```

```

00510      *
00511      * I/P :
00512      *   IX = POINTER TO START OF KEYWORD
00513      *   TBLPTR = START ADDRESS OF LOOKUP TABLE
00514      *
00515      * O/P :
00516      *   IX = START ADDRESS OF KEYWORD ROUTINE (IF FOUND)
00517      *   XTEMP1 = PTR TO CHAR FOLLOWING 3 CHAR KEYWORD
00518      *             (IF FOUND)
00519      *   ACCB = INDICATOR FOR KEYWORD FOUND
00520      *           = 00   IF NOT FOUND
00521      *           = FF   IF FOUND
00522      *
00523      * *****
00524 2A00 FF 20B3 FINDKE STX    XTEMP0    STORE START ADDR OF KEYWORD
00525 2A03 C6 05 FINDK1 LDA B    #5        SET COUNTER
00526 2A05 A6 00 FINDK2 LDA A    0,X       GET CHAR AT KEYWORD POINTER
00527 2A07 08          INX          SET KEYWORD PTR AT NEXT CHAR
00528 2A08 FF 20B5          STX    XTEMP1    STORE KEYWORD POINTER
00529 2A0B FE 20AA          LDX    TBLPTR    RETRIEVE TABLE POINTER
00530 2A0E A1 00          CMP A    0,X       COMPARE CHAR WITH TABLE
00531 2A10 27 10          BEQ    FINDK4     IF CHAR IS DIFFERENT THEN
00532 2A12 08          FINDK3 INX          MOVE POINTER
00533 2A13 5A          DEC B          TO NEXT
00534 2A14 26 FD          BNE    FINDK3     KEYWORD IN
00535 2A16 FF 20AA          STX    TBLPTR    LOOK UP TABLE
00536 2A19 6D 00          TST    0,X       AND IF END OF LOOK UP
00537 2A1B 27 16          BEQ    FINDK6     TABLE THEN EXIT OTHERWISE
00538 2A1D FE 20B3          LDX    XTEMP0    GET START OF KEYWORD AGAIN
00539 2A20 20 E2          BRA    FINDK1     AND ATTEMPT TO MATCH KEYWORD
00540 2A22 08          FINDK4 INX          IF CHAR IS SAME THEN
00541 2A23 5A          DEC B          DECREMENT COUNTER
00542 2A24 C1 02          CMP B    #2       CHECK IF ALL KEYWORD MATCHES
00543 2A26 27 08          BEQ    FINDK5     AND IF NOT
00544 2A28 FF 20AA          STX    TBLPTR    STORE TABLE POINTER
00545 2A2B FE 20B5          LDX    XTEMP1    AND RETRIEVE KEYWORD POINTER
00546 2A2E 20 D6          BRA    FINDK2     AND CHECK NEXT CHAR
00547 2A30 EE 00          FINDK5 LDX    0,X   GET ADDR OF KEYWORD ROUTINE
00548 2A32 39          RTS
00549 2A33 FE 20B3 FINDK6 LDX    XTEMP0    SET LINE POINTER BACK
00550 2A36 FF 20B5          STX    XTEMP1    IN ORIGINAL POSITION
00551 2A39 39          RTS
00552      *
00553      *      KEYWORD LOOKUP TABLES
00554      *      -----
00555      *
00556 2A3A 4C          COMTBL FCC    'LIS'    START COMMAND KEYWORD TABLE
00557 2A3B 49
00558 2A3C 53
00557 2A3D 2AC0          FDB    LIST
00558 2A3F 4D          FCC    'MON'
00559 2A40 4F

```

	2A41	4E			
00559	2A42	2B3E	FDB	MONITR	
00560	2A44	4E	FCC	'NEW'	
	2A45	45			
	2A46	57			
00561	2A47	281F	FDB	NEW	
00562	2A49	52	FCC	'RUN'	
	2A4A	55			
	2A4B	4E			
00563	2A4C	2B47	FDB	RUN	
00564	2A4E	00	FCB	\$00	END COMMAND KEYWORD TABLE
00565	2A4F	44	RUNTBL FCC	'DAT'	START RUN KEYWORD TABLE
	2A50	41			
	2A51	54			
00566	2A52	3110	FDB	REM	
00567	2A54	46	FCC	'FOR'	
	2A55	4F			
	2A56	52			
00568	2A57	2BE1	FDB	FOR	
00569	2A59	47	FCC	'GOS'	
	2A5A	4F			
	2A5B	53			
00570	2A5C	2D39	FDB	GOSUB	
00571	2A5E	49	FCC	'INP'	
	2A5F	4E			
	2A60	50			
00572	2A61	2DF2	FDB	INPUT	
00573	2A63	4E	FCC	'NEX'	
	2A64	45			
	2A65	58			
00574	2A66	2ED5	FDB	NEXT	
00575	2A68	52	FCC	'REA'	
	2A69	45			
	2A6A	41			
00576	2A6B	3088	FDB	READ	
00577	2A6D	45	IMMTBL FCC	'END'	START IMMEDIATE KEYWORD TABLE
	2A6E	4E			
	2A6F	44			
00578	2A70	2BCF	FDB	END	
00579	2A72	47	FCC	'GOT'	
	2A73	4F			
	2A74	54			
00580	2A75	2D51	FDB	GOTO	
00581	2A77	49	FCC	'IF'	
	2A78	46			
	2A79	20			
00582	2A7A	2D90	FDB	IF	
00583	2A7C	4C	FCC	'LET'	
	2A7D	45			
	2A7E	54			
00584	2A7F	2E7E	FDB	EXPRSN	
00585	2A81	50	FCC	'POK'	

	2A82 4F			
	2A83 4B			
00586	2A84 2F8A	FDB	POKE	
00587	2A86 50	FCC	'PRI'	
	2A87 52			
	2A88 49			
00588	2A89 2FBB	FDB	PRINT	
00589	2A8B 52	FCC	'REM'	
	2A8C 45			
	2A8D 4D			
00590	2A8E 3110	FDB	REM	
00591	2A90 52	FCC	'RET'	
	2A91 45			
	2A92 54			
00592	2A93 3111	FDB	RETURN	
00593	2A95 53	FCC	'STO'	
	2A96 54			
	2A97 4F			
00594	2A98 315B	FDB	STOP	
00595	2A9A 00	FCB	\$00	END RUN AND IMMEDIATE TABLES
00596	2A9B 44	IFTBL FCC	'DAT'	START "IF" KEYWORD TABLE
	2A9C 41			
	2A9D 54			
00597	2A9E 3110	FDB	REM	
00598	2AA0 46	FCC	'FOR'	(TABLE OF
	2AA1 4F			
	2AA2 52			
00599	2AA3 2BE1	FDB	FOR	
00600	2AA5 49	FCC	'IF'	KEYWORDS NOT
	2AA6 46			
	2AA7 20			
00601	2AA8 2D90	FDB	IF	
00602	2AAA 49	FCC	'INP'	ALLOWED
	2AAB 4E			
	2AAC 50			
00603	2AAD 2DF2	FDB	INPUT	
00604	2AAF 4E	FCC	'NEX'	WITHIN
	2AB0 45			
	2AB1 58			
00605	2AB2 2ED5	FDB	NEXT	
00606	2AB4 52	FCC	'REM'	IF..THEN..)
	2AB5 45			
	2AB6 4D			
00607	2AB7 3110	FDB	REM	
00608	2AB9 00	FCB	\$00	END "IF" KEYWORD TABLE
00609	2ABA 50	FNTBL FCC	'PEE'	START FUNCTION KEYWORD TABLE
	2ABB 45			
	2ABC 45			
00610	2ABD 355A	FDB	PEEK	
00611	2ABF 00	FCB	\$00	END FUNCTION KEYWORD TABLE
00612				
00613				

★
★

```

00614          *          LIST
00615          *          ----
00616          *
00617          *          THIS ROUTINE LISTS EITHER THE ENTIRE USER'S
00618          *          PROGRAM OR A SUBSET OF IT, WHERE THE SUBSET MAY BE
00619          *          ONLY ONE LINE, DEPENDING ON THE PARAMETERS
00620          *          SUPPLIED IN THE INPUT BUFFER.
00621          *
00622          *          I/P :
00623          *          XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
00624          *
00625          *****
00626 2AC0 BD 7402 LIST JSR PCRLF ADVANCE ONE LINE
00627 2AC3 FE 20B5 LDX XTEMP1 RETRIEVE SAVED BUFFER PTR
00628 2AC6 BD 207D JSR FNDSPC MOVE PTR PAST KEYWORD
00629 2AC9 81 0D CMP A #CR IF THIS CHAR IS A <CR>
00630 2ACB 26 0B BNE LIST1 THEN SET LIST POINTER
00631 2ACD CE 1001 LDX #PROGST AT START OF USER PROC
00632 2AD0 B6 2093 LDA A LRGLIN ,SET HIGHEST LINE NO.
00633 2AD3 F6 2094 LDA B LRGLIN+1 TO LARGEST CURRENT LINE NO.
00634 2AD6 20 22 BRA OUTLIS AND PRINT LISTING
00635 2AD8 BD 35E1 LIST1 JSR NUMBCD OTHERWISE READ LINE NUMBER
00636 2ADB FF 207D STX BUFPTR STORE BUFFER POINTER
00637 2ADE BD 3580 JSR FINDLN POSN LIST PTR AT LOWEST LINE
00638 2AE1 FE 207D LDX BUFPTR RETRIEVE BUFFER POINTER
00639 2AE4 BD 377A JSR SKPSPC SKIP ANY FOLLOWING SPACES
00640 2AE7 81 0D CMP A #CR IF NEXT CHARACTER IS NOT
00641 2AE9 27 03 BEQ LIST2 A <CR> THEN
00642 2AEB BD 35E1 JSR NUMBCD READ HIGHEST LINE NUMBER
00643 2AEE BD 3799 LIST2 JSR CHKCR1 CHECK FOR <CR> AT END OF LINE
00644 2AF1 FE 208D LDX FNDPTR RETRIEVE LIST POINTER
00645 2AF4 B6 2096 LIST3 LDA A MSPART SET UP HIGHEST LINE NO.
00646 2AF7 F6 2095 LDA B LSPART ( = LOWEST IF NOT SPECIFIED
00647 2AFA 0E OUTLIS CLI ENABLE <BREAK> KEY
00648 2AFB BC 20A0 CPX PROGEN IF LIST POINTER IS AT END OF
00649 2AFE 27 3C BEQ ENDLIS USER PROG, LISTING COMPLETE
00650 2B00 A1 00 CMP A 0,X IF LINE NUMBER AT
00651 2B02 22 06 BHI OUTLS1 LIST POINTER IS HIGHER
00652 2B04 26 36 BNE ENDLIS THAN HIGHEST LINE
00653 2B06 E1 01 CMP B 1,X NUMBER THEN LISTING
00654 2B08 25 32 BCS ENDLIS IS COMPLETE OTHERWISE
00655 2B0A FF 208D OUTLS1 STX FNDPTR STORE LIST POINTER
00656 2B0D B7 2096 STA A MSPART STORE HIGHEST
00657 2B10 F7 2095 STA B LSPART LINE NUMBER
00658 2B13 BD 360F JSR BCDNUM PRINT OUT CURRENT
00659 2B16 CE 2087 LDX #ERRLIN LINE'S NUMBER
00660 2B19 BD 703F JSR PSTRNG
00661 2B1C BD 70C5 JSR OUTS PRINT A SPACE
00662 2B1F FE 208D LDX FNDPTR RETRIEVE LIST POINTER
00663 2B22 08 INX INCREMENT LIST POINTER
00664 2B23 08 INX PAST LENGTH OF LINE
00665 2B24 08 OUTLS2 INX

```



```

00666 2B25 A6 00          LDA A 0,X      READ CHAR AT LIST POINTER
00667 2B27 BD 70E7        JSR OUT        PRINT CHAR AT LIST POINTER
00668 2B2A 81 0D          CMP A #CR      IF CHAR WAS NOT A <CR>
00669 2B2C 26 F7          BNE OUTL$2     THEN GET NEXT CHARACTER
00670 2B2E 86 0A          LDA A #LF      OTHERWISE PRINT A <LF>
00671 2B30 BD 70E7        JSR OUT
00672 2B33 08             INX             AND REPEAT FOR NEXT LINE
00673 2B34 7D 207C        TST BRKFLG    IF <BREAK> WAS
00674 2B37 27 BC          BEQ LIST3     PRESSED THEN GO AND
00675 2B39 7E 3156        JMP BREAK     PROCESS <BREAK>
00676 2B3C 0F             ENDLIS SEI      DISABLE <BREAK> KEY
00677 2B3D 39             RTS
00678                      *
00679                      *
00680                      *          MONITOR
00681                      *          -----
00682                      *
00683                      *          THIS ROUTINE RETURNS CONTROL TO THE DATERM
00684                      *          MONITOR PROGRAM.
00685                      *
00686                      *          I/P :
00687                      *          XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
00688                      *
00689                      *          *****
00690 2B3E FE 20B5 MONITR LDX XTEMP1 RETRIEVE SAVED BUFFER PTR
00691 2B41 BD 3799        JSR CHKCR1     CHECK FOR <CR> AFTER KEYWORD
00692 2B44 7E 700B        JMP RESET     RETURN TO MONITOR
00693                      *
00694                      *
00695                      *          RUN
00696                      *          ---
00697                      *
00698                      *          THIS ROUTINE RUNS AN ENTIRE USER PROGRAM
00699                      *          STARTING AT THE FIRST LINE IN MEMORY UNLESS "STOP"
00700                      *          OR "END" ARE ENCOUNTERED, OR THE <BREAK> KEY IS
00701                      *          PRESSED.
00702                      *
00703                      *          I/P :
00704                      *          XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
00705                      *
00706                      *          *****
00707 2B47 FE 20B5 RUN      LDX XTEMP1 RETRIEVE SAVED BUFFER PTR
00708 2B4A BD 3799        JSR CHKCR1     CHECK FOR <CR> AFTER KEYWORD
00709 2B4D CE 1001        LDX #PROGST    GET PROGRAM START ADDR
00710 2B50 BC 20A0        CPX PROGEN     IF NO PROGRAM IN MEMORY
00711 2B53 27 3A          BEQ RUN3      THEN END RUN OTHERWISE
00712 2B55 CE 1000        LDX #PROGST-1 SET UP DATA PTR AT <CR>
00713 2B58 FF 2080        STX DATPTR    IMMEDIATELY BEFORE USER PROG
00714 2B5B CE 2000        LDX #VARBEG
00715 2B5E DF 0E          STX STARAD    SET ALL USER
00716 2B60 CE 2033        LDX #VAREND
00717 2B63 DF 10          STX ENDADD    VARIABLES

```

```

00718 2B65 4F          CLR A
00719 2B66 BD 722F     JSR    FILMEM    TO ZERO
00720 2B69 CE 27FF     LDX    #USRSTK  SET USER STACK PTR
00721 2B6C FF 20AF     STX    USRSP    TO TOP OF USER STACK
00722 2B6F CE 1001     LDX    #PROGST  INITIALISE RUN PTR AT FIRST
00723 2B72 FF 20A4     STX    RUNPTR   INSTRUCTION OF USER PROG
00724 2B75 FE 1001     LDX    PROGST  SET PROGRAM COUNTER = FIRST
00725 2B78 FF 209E     STX    PRGCNT   LINE NO. OF USER PROGRAM
00726 2B7B 0E          GORUN CLI      ENABLE <BREAK> KEY
00727 2B7C 8D 12      RUN1 BSR      EXECUT  EXECUTE A PROGRAM LINE
00728 2B7E 7D 207C     TST    BRKFLG  IF <BREAK> WAS PRESSED
00729 2B81 27 03       BEQ    RUN2    THEN GO AND
00730 2B83 BD 3156     JSR    BREAK   PROCESS <BREAK>
00731 2B86 FE 209E     LDX    PRGCNT  IF NOT END OF
00732 2B89 8C FFFF     CPX    #$FFFF  USER PROG THEN
00733 2B8C 26 EF       BNE    RUN1    EXECUTE NEXT PROGRAM LINE
00734 2B8E 0F          SET      OTHERWISE DISABE <BREAK> KEY
00735 2B8F 39          RUN3 RTS      AND END RUN
00736                *
00737                *
00738                *      EXECUTE ONE PROGRAM LINE
00739                *      -----
00740                *
00741                *      THIS ROUTINE INCREMENTS THE USER'S PROGRAM
00742                *      COUNTER AND RUN POINTER TO THE NEXT LINE OF THE
00743                *      USER'S PROGRAM THEN EXECUTES THE CURRENT PROGRAM
00744                *      LINE.
00745                *
00746                *      *****
00747 2B90 FE 209E     EXECUT LDX    PRGCNT  STORE PROGRAM COUNTER
00748 2B93 FF 2091     STX    LINENO  AS CURRENT LINE NO.
00749 2B96 FE 20A4     LDX    RUNPTR   SET LINE PTR AT
00750 2B99 FF 20B5     STX    XTEMP1  RUN PTR POSITION
00751 2B9C E6 02       LDA B 2,X      SET RUN PTR
00752 2B9E BD 375D     JSR    IXADD   AT NEXT LINE
00753 2BA1 FF 20A4     STX    RUNPTR   TO BE RUN
00754 2BA4 BC 20A0     CPX    PROGEN  IF NOT END OF USER
00755 2BA7 27 04       BEQ    EXECU1  PROG THEN INCREMENT PROGRAM
00756 2BA9 EE 00       LDX    0,X     COUNTER TO NEXT LINE
00757 2BAB 20 03       BRA    EXECU2  OTHERWISE SET PROGRAM
00758 2BAD CE FFFF     EXECU1 LDX    #$FFFF COUNTER = FFFF
00759 2BB0 FF 209E     EXECU2 STX    PRGCNT
00760 2BB3 CE 2A4F     EXECU3 LDX    #RUNTBL  SET UP TABLE POINTER AT
00761 2BB6 FF 20AA     STX    TBLPTR  START OF RUN KEYWORD TABLE
00762 2BB9 FE 20B5     LDX    XTEMP1  RETRIEVE PTR TO PROG LINE
00763 2BBC 08          INX          MOVE LINE POINTER
00764 2BBD 08          INX          PAST LINE NO.
00765 2BBE 08          INX          AND LENGTH BYTES
00766 2BBF FF 20B5     STX    XTEMP1  SAVE LINE PTR
00767 2BC2 BD 2A00     EXECU4 JSR    FINDKE  ATTEMPT TO FIND KEYWORD
00768 2BC5 5D          TST B        IF KEYWORD NOT FOUND THEN
00769 2BC6 26 04       BNE    EXECU5

```

```

00770 2BC8 BD 2E7E      JSR      EXPRSN   PROCESS EXPRESSION
00771 2BCB 39           RTS
00772 2BCC AD 00      EXECUS JSR      0,X      OTHERWISE EXECUTE KEYWORD
00773 2BCE 39           RTS      NEXT PROG LINE
00774                *
00775                *
00776                *          END
00777                *          ---
00778                *
00779                *      THIS ROUTINE CLEARS THE USER STACK THEN RETURNS
00780                *      TO THE INTERPRETER PROCESS, ENDING THE RUN OF A
00781                *      USER PROGRAM.
00782                *
00783                *      I/P :
00784                *      XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
00785                *
00786                *      *****
00787 2BCF FE 20B5      END      LDX      XTEMP1   RETRIEVE SAVED LINE PTR
00788 2BD2 BD 37A3      JSR      CHKCR2   CHECK FOR <CR> AFTER KEYWORD
00789 2BD5 CE 27FF      END1    LDX      #USRSTK   CLEAR USER'S STACK
00790 2BD8 FF 20AF      STX      USRSP
00791 2BDB BE 20A7      END2    LDS      SP      CLEAR SYSTEM STACK
00792 2BDE 7E 284E      JMP      INTPRT   EXECUTE INTERPRETER PROCESS
00793                *
00794                *
00795                *          FOR...TO...(STEP)
00796                *          -----
00797                *
00798                *      THIS ROUTINE INITIATES A BASIC PROGRAM LOOP BY
00799                *      ASSIGNING AN INITIAL AND A FINAL VALUE TO A
00800                *      VARIABLE AND BY ALSO DEFINING THE STEP SIZE BY
00801                *      WHICH THE VARIABLE SHALL BE INCREMENTED/
00802                *      DECREMENTED UNTIL IT REACHES ITS FINAL VALUE.
00803                *
00804                *      I/P :
00805                *      XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
00806                *
00807                *      *****
00808 2BE1 FE 20B5      FOR      LDX      XTEMP1   RETRIEVE LINE PTR
00809 2BE4 BD 377A      JSR      SKPSPC   MOVE PTR TO FIRST NONSPACE
00810 2BE7 BD 3746      JSR      CLASS    CLASSIFY CHARACTER
00811 2BEA C1 02      CMP      B      #$02   IF CHAR IS NOT A
00812 2BEC 27 03      BEQ      FOR2     VARIABLE NAME THEN
00813 2BEE 7E 319E      FOR1    JMP      SYNERR   SYNTAX ERROR OTHERWISE :
00814 2BF1 A6 00      FOR2    LDA      A      0,X   READ AND
00815 2BF3 B7 20B2      STA      A      VARNAM   STORE VARIABLE NAME
00816 2BF6 08          INX
00817 2BF7 BD 377A      JSR      SKPSPC   MOVE LINE PTR PAST VARIABLE
00818 2BFA 86 3D      LDA      A      #'=   IF CHAR IS NOT "="
00819 2BFC A1 00      CMP      A      0,X   THEN
00820 2BFE 26 EF      BNE      FOR1     SYNTAX ERROR OTHERWISE :
00821 2C00 FF 20B5      STX      XTEMP1   SAVE LINE PTR

```

00822	2C03	FE 20AF	LDX	USRSP	SET UP TEMP USER SP
00823	2C06	7F 208C	CLR	FLAG	INDICATE NO GOSUBS ON STACK
00824	2C09	8C 27FF FOR3	CPX	#USRSTK	IF AT END OF STACK OR
00825	2C0C	27 26	BEQ	FOR7	
00826	2C0E	A6 01	LDA A	1,X	IF VARIABLE NAME IS ALREADY
00827	2C10	B1 2082	CMP A	VARNAM	
00828	2C13	27 0E	BEQ	FOR5	ON STACK THEN EXIT LOOP
00829	2C15	81 00	CMP A	#\$00	IF GOSUB ADDR FOUND
00830	2C17	26 05	BNE	FOR4	THEN INDICATE GOSUB
00831	2C19	86 FF	LDA A	#\$FF	ADDRESS EXISTS
00832	2C1B	B7 208C	STA A	FLAG	ON STACK
00833	2C1E	08 FOR4	INX		
00834	2C1F	08	INX		DECREMENT TEMP STACK PTR
00835	2C20	08	INX		
00836	2C21	20 E7	BRA	FOR3	REPEAT LOOP
00837	2C23	7D 208C FOR5	TST	FLAG	IF GOSUB ADDR FOUND
00838	2C26	27 06	BEQ	FOR6	ON USER STACK THEN
00839	2C28	CE 4752	LDX	#GOSRET	GOSUB WITHOUT RETURN ERROR
00840	2C2B	7E 31A1	JMP	ERROR	
00841	2C2E	08 FOR6	INX		
00842	2C2F	08	INX		DECREMENT TEMP STACK PTR
00843	2C30	08	INX		
00844	2C31	FF 20AF	STX	USRSP	STORE TEMP STACK PTR
00845	2C34	FE 20AF FOR7	LDX	USRSP	GET ORIGINAL STACK PTR
00846	2C37	BD 3784	JSR	CHKSTK	ENSURE NO STACK OVERFLOW
00847	2C3A	B6 2092	LDA A	LINENO+1	PUSH CURRENT LINE
00848	2C3D	A7 00	STA A	0,X	
00849	2C3F	09	DEX		NUMBER AND VARIABLE
00850	2C40	B6 2091	LDA A	LINENO	
00851	2C43	A7 00	STA A	0,X	NAME ONTO USER STACK
00852	2C45	09	DEX		
00853	2C46	B6 2082	LDA A	VARNAM	AND INCREMENT USER'S
00854	2C49	A7 00	STA A	0,X	
00855	2C4B	09	DEX		STACK POINTER
00856	2C4C	FF 20AF	STX	USRSP	
00857	2C4F	FE 20B5	LDX	XTEMP1	RETRIEVE LINE PTR
00858	2C52	08	INX		MOVE LINE PTR PAST "="
00859	2C53	BD 377A	JSR	SKPSPC	MOVE POINTER TO NEXT NONSPACE
00860	2C56	BD 31E0	JSR	EVAL	EVALUATE EXPRESSION
00861	2C59	BD 2EC1	JSR	EXPR55	ASSIGN RESULT TO VARIABLE
00862	2C5C	8D 16	BSR	TO	PROCESS TO, STEP STMTS
00863	2C5E	BD 35C9	JSR	GETVAR	GET VARIABLE VALUE
00864	2C61	7D 209C	TST	OPRND1	IF STEP VALUE IS POSITIVE
00865	2C64	2B 06	BMI	FOR10	AND VARIABLE VALUE
00866	2C66	BD 2F71	JSR	DUDCPX	IS > END LOOP VALUE
00867	2C69	2E 06	BGT	FOR11	OR
00868	2C6B	39 FOR9	RTS		IF STEP VALUE IS NEGATIVE
00869	2C6C	BD 2F71 FOR10	JSR	DUDCPX	AND VARIABLE VALUE
00870	2C6F	2C FB	BGE	FOR9	IS < END LOOP VALUE
00871	2C71	8D 68 FOR11	BSR	ENDFOR	THEN END OF FOR LOOP
00872	2C73	39	RTS		OTHERWISE CONTINUE NEXT LINE
00873		*			

```

00874      *
00875      *          TO
00876      *          --
00877      *
00878      *      THIS ROUTINE DETERMINES THE TERMINATING VALUE
00879      *      FOR A LOOP VARIABLE FROM THE FOR...TO...(STEP)
00880      *      STATEMENT.
00881      *
00882      *      I/P :
00883      *      XTEMP1 = LINE POINTER
00884      *
00885      *      O/P :
00886      *      XTEMP1 = LINE POINTER (UPDATED)
00887      *      ENDVAL = TERMINATING VALUE FOR LOOP VARIABLE
00888      *
00889      *      *****
00890 2C74 FE 20B5 TO      LDX      XTEMP1      RETRIEVE LINE PTR
00891 2C77 A6 00          LDA A    0,X          CHECK THAT
00892 2C79 E6 01          LDA B    1,X
00893 2C7B 81 54          CMP A    #'T          NEXT TWO CHARS
00894 2C7D 27 03          BEQ      TO2
00895 2C7F 7E 319E TO1    JMP      SYNERR      ARE "TO" - IF NOT
00896 2C82 C1 4F TO2     CMP B    #'O
00897 2C84 26 FA          BNE      TO1          THEN SYNTAX ERROR
00898 2C86 08             INX              MOVE LINE PTR PAST
00899 2C87 08             INX              KEYWORD "TO"
00900 2C88 BD 377A        JSR      SKSPSC      MOVE PTR TO NEXT NONSPACE
00901 2C8B BD 31E0        JSR      EVAL        EVALUATE EXPRESSION
00902 2C8E FE 20A2        LDX      RESULT      ASSIGN RESULT TO
00903 2C91 FF 2082        STX      ENDVAL      END LOOP VARIABLE
00904      *
00905      *
00906      *          STEP
00907      *          ----
00908      *
00909      *      THIS ROUTINE DETERMINES THE STEPPING VALUE FOR
00910      *      A LOOP VARIABLE FROM THE FOR...TO...(STEP)
00911      *      STATEMENT.
00912      *
00913      *      I/P :
00914      *      XTEMP1 = LINE POINTER
00915      *
00916      *      O/P :
00917      *      OPRND1 = STEP VALUE FOR LOOP VARIABLE
00918      *
00919      *      *****
00920 2C94 FE 20B5 STEP    LDX      XTEMP1      RETRIEVE LINE PTR
00921 2C97 BD 377A        JSR      SKSPSC      MOVE PTR TO NEXT NONSPACE
00922 2C9A 81 0D          CMP A    #CR        IF NO STEP IS
00923 2C9C 26 09          BNE      STEP1      SPECIFIED THEN
00924 2C9E 7F 209C        CLR      OPRND1
00925 2CA1 86 01          LDA A    #$01        SET STEP VALUE TO +1

```

```

00926 2CA3 B7 209D      STA A   OPRND1+1
00927 2CA6 39          RTS
00928 2CA7 FF 20B5 STEP1 STX      XTEMP1  OTHERWISE :
                                STORE LINE POINTER
00929 2CAA EE 00          LDX      0,X
00930 2CAC 8C 5354        CPX      #$5354  CHECK THAT
00931 2CAF 27 03          BEQ      STEP3
00932 2CB1 7E 319E STEP2 JMP      SYNERR  NEXT STATEMENT
00933 2CB4 FE 20B5 STEP3 LDX      XTEMP1
00934 2CB7 EE 02          LDX      2,X      IS "STEP"
00935 2CB9 8C 4550        CPX      #$4550
00936 2CBC 26 F4          BNE      STEP2
00937 2CBE FE 20B5        LDX      XTEMP1  RETRIEVE LINE PTR
00938 2CC1 08            INX
00939 2CC2 08            INX      MOVE LINE PTR PAST
00940 2CC3 08            INX
00941 2CC4 08            INX      "STEP" STATEMENT
00942 2CC5 BD 377A        JSR      SKSPSPC  MOVE PTR TO NEXT NONSPACE
00943 2CC8 BD 31E0        JSR      EVAL    EVALUATE EXPRESSION
00944 2CCB FE 20B5        LDX      XTEMP1  RETRIEVE LINE PTR
00945 2CCE BD 37A3        JSR      CHKCR2  CHECK FOR <CR> AT END OF LINE
00946 2CD1 FF 20B5        STX      XTEMP1  STORE LINE POINTER
00947 2CD4 FE 20A2        LDX      RESULT  GET STEP VALUE AND
00948 2CD7 FF 209C        STX      OPRND1  STORE IN OPRND1
00949 2CDA 39            RTS
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965 2CDB FE 20AF ENDFOR LDX      USRSP    REMOVE LOOP REFERENCE
00966 2CDE 08            INX
00967 2CDF 08            INX      FROM USER STACK
00968 2CE0 08            INX
00969 2CE1 FF 20AF        STX      USRSP
00970 2CE4 FE 20A4        LDX      RUNPTR   RETRIEVE TEMP RUN PTR
00971 2CE7 BC 20A0 ENDFR1 CPX      PROGEN  COMPARE PTR TO PROG END
00972 2CEA 27 2E          BEQ      ENDFR3  IF PTR NOT AT PROG END THEN:
00973 2CEC A6 03          LDA A   3,X
00974 2CEE 81 4E          CMP A   #'N      COMPARE FIRST THREE
00975 2CF0 26 1B          BNE      ENDFR2
00976 2CF2 A6 04          LDA A   4,X      CHARACTERS OF TEMP RUN PTR
00977 2CF4 81 45          CMP A   #'E

```

```

00978 2CF6 26 15      BNE   ENDFR2   LINE WITH "NEX". IF
00979 2CF8 A6 05      LDA A   5,X
00980 2CFA 81 58      CMP A   #'X     CHARS = "NEX" THEN
00981 2CFC 26 0F      BNE   ENDFR2
00982 2CFE C6 05      LDA B   #5     MOVE TEMP RUN PTR
00983 2D00 BD 375D     JSR     IXADD
00984 2D03 BD 376D     JSR     FNDSPC   PAST "NEXT" STATEMENT AND
00985 2D06 A6 00      LDA A   0,X     COMPARE VARIABLE FOUND
00986 2D08 B1 20B2     CMP A   VARNAM WITH FOR LOOP VARIABLE. IF
00987 2D0B 27 13      BEQ     ENDFR4   CORRECT VAR THEN STOP SEARCH
00988 2D0D FE 20A4     LDX     RUNPTR  OTHERWISE RETRIEVE RUN PTR,
00989 2D10 E6 02      LDA B   2,X
00990 2D12 BD 375D     JSR     IXADD   ADD LENGTH OF LINE TO IT
00991 2D15 FF 20A4     STX     RUNPTR  AND REPLACE IT
00992 2D18 20 CE      BRA     ENDFR1  THEN CONTINUE SEARCH
00993 2D1A CE 464E     LDX     #FORNEX IF RUN PTR AT END OF PROG
00994 2D1D BD 31A1     JSR     ERROR  THEN FOR WITHOUT NEXT ERROR
00995 2D20 FE 20A4     LDX     RUNPTR  CORRECT "NEXT" FOUND :
00996 2D23 E6 02      LDA B   2,X
00997 2D25 BD 375D     JSR     IXADD
00998 2D28 BC 20A0     CPX     PROGEN  IF AT END OF PROGRAM
00999 2D2B 26 03      BNE     ENDFR6   THEN END, OTHERWISE
01000 2D2D 7E 2BD5     JMP     ENDL   SET RUN PTR AT ADDR OF
01001 2D30 FF 20A4     STX     RUNPTR  LINE FOLLOWING NEXT STMT
01002 2D33 EE 00      LDX     0,X     AND SET PROG COUNTER TO
01003 2D35 FF 209E     STX     PRGCNT  THE LINE NO. OF THAT LINE
01004 2D38 39          RTS
01005                  *
01006                  *
01007                  *      GOSUB
01008                  *      -----
01009                  *
01010                  *      THIS ROUTINE TRANSFERS CONTROL TEMPORARILY TO A
01011                  *      SUBROUTINE STARTING AT A SPECIFIED LINE NUMBER.
01012                  *      THE RETURN-TO LINE NUMBER (I.E. THE NEXT LINE NO.
01013                  *      IN SEQUENCE) IS PLACED ON THE USER STACK AND THEN
01014                  *      CONTROL WITHIN THE USER'S PROGRAM IS TRANSFERRED
01015                  *      TO THE SUBROUTINE USING GOTO.
01016                  *
01017                  *      I/P :
01018                  *      XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
01019                  *
01020                  *      *****
01021 01021 2D39 BD 3784     GOSUB  JSR     CHKSTK  ENSURE NO STACK OVERFLOW
01022 01022 2D3C FE 20AF     LDX     USRSP    THEN PUSH THE
01023 01023 2D3F B6 209F     LDA A   PRGCNT+1
01024 01024 2D42 A7 00      STA A   0,X     RETURN LINE NUMBER
01025 01025 2D44 09          DEX
01026 01026 2D45 B6 209E     LDA A   PRGCNT  ONTO THE USER'S STACK
01027 01027 2D48 A7 00      STA A   0,X
01028 01028 2D4A 09          DEX
01029 01029 2D4B 6F 00      CLR     0,X     AND INCREMENT THE

```

```

01030 2D4D 09          DEX          USER'S STACK POINTER
01031 2D4E FF 20AF    STX          USRSP
01032                *
01033                *
01034                *          GOTO
01035                *          ----
01036                *
01037                *          THIS ROUTINE TRANSFERS CONTROL TO ANOTHER LINE
01038                *          NUMBER BY READING THE LINE NUMBER THEN UPDATING
01039                *          THE PROGRAM COUNTER TO THIS LINE NUMBER AND ALSO
01040                *          UPDATING THE RUN POINTER TO POINT TO THE NEW LINE
01041                *
01042                * I/P :
01043                *      XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
01044                *
01045                * *****
01046 2D51 FE 20B5 GOTO   LDX          XTEMP1    RETRIEVE LINE PTR
01047 2D54 BD 376D      JSR          FNDSPC     MOVE PTR PAST KEYWORD
01048 2D57 BD 3746      JSR          CLASS     CLASSIFY FIRST NONSPACE
01049 2D5A C1 01      CMP B        #$01      IF THIS CHAR IS
01050 2D5C 27 03        BEQ          GOTO1     NOT NUMERIC THEN
01051 2D5E 7E 319E      JMP          SYNERR    DISPLAY SYNTAX ERROR MESSAGE
01052 2D61 BD 35E1 GOTO1 JSR          NUMBCD    OTHERWISE READ LINE NUMBER
01053 2D64 BD 37A3      JSR          CHKCR2   CHECK FOR <CR> AT END OF LINE
01054 2D67 BD 3580 GOTO2 JSR          FINDLN   FIND LINE IN USER PROG
01055 2D6A 5D          TST B              IF LINE WANT TO GO TO
01056 2D6B 27 06        BEQ          GOTO3     IS NONEXISTENT THEN
01057 2D6D CE 4E4C      LDX          #NOLINE   SET UP ERROR CODE
01058 2D70 7E 31A1      JMP          ERROR    DISPLAY ERROR MESSAGE
01059 2D73 B6 2096 GOTO3 LDA A        MSPART   OTHERWISE STORE LINE NO.
01060 2D76 B7 209E      STA A        PRGCNT
01061 2D79 B6 2095      LDA A        LSPART    IN PROGRAM COUNTER AND
01062 2D7C B7 209F      STA A        PRGCNT+1
01063 2D7F FF 20A4      STX          RUNPTR   SET UP RUN PTR AT NEXT LINE
01064 2D82 7D 2091      TST          LINENO   IF GOTO USED
01065 2D85 26 08        BNE          GOTO4    FROM IMMEDIATE MODE
01066 2D87 7D 2092      TST          LINENO+1 THEN EXECUTE
01067 2D8A 26 03        BNE          GOTO4    PROGRAM FROM
01068 2D8C BD 2B7B      JSR          GORUN     GOTO LINE NUMBER OTHERWISE
01069 2D8F 39          GOTO4 RTS          RETURN
01070                *
01071                *
01072                *          IF...THEN
01073                *          -----
01074                *
01075                *          THIS ROUTINE PROCESSES THE BASIC STATEMENT
01076                *          FOLLOWING THE THEN KEYWORD ONLY ON FINDING THE
01077                *          TRUE RESULT OF EVALUATING AN EXPRESSION. IF THE
01078                *          EXPRESSION RESULT IS FALSE, NO FURTHER PROCESSING
01079                *          IS CARRIED OUT.
01080                *
01081                * I/F :

```



```

01082          * XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
01083          *
01084          *****
01085 2D90 FE 20B5 IF      LDX      XTEMP1  RETRIEVE LINE POINTER
01086 2D93 BD 377A        JSR      SKSPSPC MOVE PTR TO FIRST NONSPACE
01087 2D96 BD 31E0        JSR      EVAL    EVALUATE EXPRSN
01088 2D99 FE 20B5        LDX      XTEMP1  RETRIEVE LINE POINTER
01089 2D9C EE 00          LDX      0,X
01090 2D9E 8C 5448        CPX      #$5448  CHECK THAT
01091 2DA1 26 0A          BNE      IF1
01092 2DA3 FE 20B5        LDX      XTEMP1  NEXT STATEMENT
01093 2DA6 EE 02          LDX      2,X
01094 2DA8 8C 454E        CPX      #$454E  IS "THEN"
01095 2DAB 27 03          BEQ      IF2
01096 2DAD 7E 319E IF1   JMP      SYNERR  DISPLAY SYNTAX ERROR MESSAGE
01097 2DB0 FE 20B5 IF2   LDX      XTEMP1  RETRIEVE LINE PTR
01098 2DB3 08            INX
01099 2DB4 08            INX      MOVE LINE PTR
01100 2DB5 08            INX
01101 2DB6 08            INX
01102 2DB7 FF 20B5        STX      XTEMP1  STORE LINE PTR
01103 2DBA 7D 20A2        TST      RESULT IF RESULT OF
01104 2DBD 26 06          BNE      THEN  EVALUATING EXPRESSION
01105 2DBF 7D 20A3        TST      RESULT+1 IS ZERO THEN
01106 2DC2 26 01          BNE      THEN
01107 2DC4 39            RTS      NO FURTHER PROCESSING
01108 2DC5 BD 377A THEN  JSR      SKSPSPC OTHERWISE SKIP TO NONSPACE
01109 2DC8 FF 20B5        STX      XTEMP1  STORE LINE POINTER
01110 2DCB CE 2A9B        LDX      #IFTLBL IF THE STATEMENT
01111 2DCE FF 20AA        STX      TBLPTR
01112 2DD1 FE 20B5        LDX      XTEMP1  FOLLOWING "THEN" IS AN
01113 2DD4 BD 2A00        JSR      FINDKE
01114 2DD7 5D            TST      B      ILLEGAL ONE FOR IF..THEN,
01115 2DD8 27 02          BEQ      THEN1
01116 2DDA 20 D2          BRA      IF1      THEN SYNTAX ERROR OTHERWISE
01117 2DDC FE 2091 THEN1 LDX      LINENO  SET TABLE PTR AT THE
01118 2DDF 27 05          BEQ      THEN2
01119 2DE1 CE 2A4F        LDX      #RUNTBL  APPROPRIATE LOOKUP TABLE
01120 2DE4 20 03          BRA      THEN3
01121 2DE6 CE 2A6D THEN2 LDX      #IMMTBL  AS DETERMINED BY THE
01122 2DE9 FF 20AA THEN3 STX      TBLPTR  CURRENT MODE
01123 2DEC FE 20B5        LDX      XTEMP1
01124 2DEF 7E 2BC2        JMP      EXECU4  EXECUTE STATEMENT
01125          *
01126          *
01127          *      INPUT
01128          *      -----
01129          *
01130          *      THIS ROUTINE ASSIGNS A VALUE TO EACH VARIABLE
01131          *      IN THE VARIABLE LIST WHICH FOLLOWS THE KEYWORD,
01132          *      WHERE THE VALUES ARE ENTERED BY THE USER IN AN
01133          *      INPUT BUFFER.

```

```

01134          *
01135          * I/P :
01136          *   XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
01137          *
01138          *****
01139 2DF2 FE 20B5 INPUT LDX XTEMP1 RETRIEVE LINE POINTER
01140 2DF5 FF 20B7 STX XTEMP2 SAVE INITIAL LINE PTR
01141 2DF8 BD 376D JSR FNDSPC MOVE PTR PAST KEYWORD
01142 2DFB BD 3746 INPUT1 JSR CLASS IF CHAR IS NOT ALPHABETIC
01143 2DFE C1 02 CMP B #$02
01144 2E00 27 03 BEQ INPUT3
01145 2E02 7E 319E INPUT2 JMP SYNERR DISPLAY SYNTAX ERROR MESSAGE
01146 2E05 A6 00 INPUT3 LDA A 0,X OTHERWISE :
01147 2E07 B7 20B1 STA A VAR STORE VARIABLE NAME
01148 2E0A FF 20B5 STX XTEMP1 STORE LINE POINTER
01149 2E0D CE 773A INPUT4 LDX #MSG2
01150 2E10 BD 703F JSR PSTRNG DISPLAY INPUT PROMPT
01151 2E13 FE 209E LDX PRGCNT SET UP PROGRAM COUNTER
01152 2E16 FF 20B3 STX XTEMP0 TO THE CURRENT LINE NO.
01153 2E19 FE 2091 LDX LINENO IN CASE <BREAK> IS PRESSED
01154 2E1C FF 209E STX PRGCNT SO INPUT REPEATED IF RETURN
01155 2E1F BD 2879 JSR FILBUF GET USER INPUT
01156 2E22 FE 20B3 LDX XTEMP0 RESTORE PROGRAM COUNTER
01157 2E25 FF 209E STX PRGCNT TO FOLLOWING LINE NO. AGAIN
01158 2E28 CE 2034 LDX #BUFFER SET PTR AT START INPUT BUFFER
01159 2E2B BD 377A INPUT5 JSR SKSPSPC MOVE PTR TO NEXT NONSPACE
01160 2E2E 81 0D CMP A #CR IF NONSPACE IS A <CR>
01161 2E30 27 DC BEQ INPUT4 THEN GO AND GET MORE INPUT
01162 2E32 7F 208C CLR FLAG SET FLAG TO INDICATE THAT
01163 2E35 73 208C COM FLAG ASCHEX CALLED FROM INPUT
01164 2E38 BD 36B2 JSR ASCHEX READ AND STORE INPUT NO.
01165 2E3B BD 377A JSR SKSPSPC MOVE PTR TO NEXT NONSPACE
01166 2E3E 81 0D CMP A #CR IF THE NEXT NONSPACE IS NOT
01167 2E40 27 0B BEQ INPUT7
01168 2E42 81 2C CMP A #', A <CR> OR ", " THEN BAD INPUT
01169 2E44 27 03 BEQ INPUT6
01170 2E46 7E 31D1 JMP BADINP
01171 2E49 08 INPUT6 INX IF NEXT NONSPACE WAS A
01172 2E4A BD 377A JSR SKSPSPC COMMA THEN SKIP COMMA
01173 2E4D FF 207D INPUT7 STX BUFPTR STORE BUFFER PTR
01174 2E50 BD 2EC7 JSR PUTVAL ASSIGN INPUT NO. TO VARIABLE
01175 2E53 FE 20B5 LDX XTEMP1 RETRIEVE LINE PTR
01176 2E56 08 INX MOVE LINE PTR PAST VAR
01177 2E57 BD 377A JSR SKSPSPC MOVE PTR TO NEXT NONSPACE
01178 2E5A 81 0D CMP A #CR IF NONSPACE CHAR IS <CR>
01179 2E5C 27 1C BEQ INPUT8 THEN FINISHED
01180 2E5E 81 2C CMP A #', OTHERWISE IF NOT A ", "
01181 2E60 26 A1 BNE INPUT2 THEN SYNTAX ERROR
01182 2E62 08 INX OTHERWISE SKIP COMMA
01183 2E63 BD 377A JSR SKSPSPC MOVE PTR TO NEXT NONSPACE
01184 2E66 BD 3746 JSR CLASS CLASSIFY NEXT NONSPACE CHAR
01185 2E69 C1 02 CMP B #$02 IF CHAR NOT ALPHABETIC

```

```

01186 2E6B 26 96      BNE  INPUT2  THEN SYNTAX ERROR
01187 2E6D A6 00      LDA  A 0,X    OTHERWISE
01188 2E6F B7 20B1    STA  A VAR    STORE VARIABLE NAME
01189 2E72 FF 20B5    STX   XTEMP1  STORE LINE POINTER
01190 2E75 FE 207D    LDX   BUFPTR  RETRIEVE BUFFER POINTER
01191 2E78 20 B2      BRA   INPUT5  REPEAT FOR NEXT VARIABLE
01192 2E7A BD 7402    INPUT8 JSR   PCRLF  IN INPUT LIST
01193 2E7D 39          RTS

01194 *
01195 *
01196 *      ASSIGN EXPRESSION (LET)
01197 *      -----
01198 *
01199 *      THIS ROUTINE ASSIGNS THE EVALUATED RESULT OF AN
01200 *      EXPRESSION TO A VARIABLE.
01201 *
01202 *      I/P :
01203 *      XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
01204 *
01205 *****
01206 2E7E FE 20B5    EXPRSN LDX   XTEMP1  RETRIEVE LINE PTR
01207 2E81 BD 377A    JSR   SKPSPC  MOVE PTR TO FIRST NONSPACE
01208 2E84 BD 3746    JSR   CLASS   CLASSIFY CHARACTER
01209 2E87 C1 02     CMP  B #02     IF CHAR IS A VARIABLE THEN
01210 2E89 26 0F      BNE  EXPRS1
01211 2E8B A6 00      LDA  A 0,X    STORE VARIABLE NAME
01212 2E8D B7 20B2    STA  A VARNAM
01213 2E90 08        INX          MOVE PTR PAST VARIABLE
01214 2E91 BD 377A    JSR   SKPSPC  MOVE PTR TO NEXT NONSPACE
01215 2E94 86 3D      LDA  A #'=    IF CHAR IS NOT "="
01216 2E96 A1 00      CMP  A 0,X    THEN
01217 2E98 27 0B      BEQ  EXPRS3
01218 2E9A FE 2091    EXPRS1 LDX   LINENO  IF CURRENTLY IN
01219 2E9D 27 03      BEQ  EXPRS2  RUN MODE THEN
01220 2E9F 7E 319E    JMP   SYNERR  DISPLAY SYNTAX ERROR MESSAGE
01221 2EA2 7E 31C5    EXPRS2 JMP   WHAT  OTHERWISE DISPLAY "WHAT?"
01222 2EA5 08        EXPRS3 INX          ELSE MOVE PTR PAST "="
01223 2EA6 BD 377A    JSR   SKPSPC  MOVE PTR TO NEXT NONSPACE
01224 2EA9 BD 31E0    JSR   EVAL    GO AND EVALUATE EXPRESSION
01225 2EAC FE 20B5    LDX   XTEMP1  RETRIEVE LINE POINTER
01226 2EAF 7D 2091    TST   LINENO
01227 2EB2 26 0A      BNE  EXPRS4  CHECK FOR <CR> AT END
01228 2EB4 7D 2092    TST   LINENO+1
01229 2EB7 26 05      BNE  EXPRS4  OF LINE DEPENDING
01230 2EB9 BD 3799    JSR   CHKCR1
01231 2EBC 20 03      BRA   EXPRS5  ON CURRENT MODE
01232 2EBE BD 37A3    EXPRS4 JSR   CHKCR2
01233 2EC1 B6 20B2    EXPRS5 LDA  A VARNAM  GET VARIABLE NAME AND
01234 2EC4 B7 20B1    STA  A VAR    STORE IT IN VAR
01235 2EC7 BD 35D4    PUTVAL JSR   GETVA1  GET VARIABLE ADDRESS
01236 2ECA B6 20A2    LDA  A RESULT  STORE MS BYTE OF RESULT
01237 2ECD A7 00      STA  A 0,X    IN VARIABLE AREA

```

```

01238 2ECF B6 20A3      LDA A RESULT+1   STORE LS BYTE OF RESULT
01239 2ED2 A7 01        STA A 1,X       IN VARIABLE AREA
01240 2ED4 39           RTS
01241                  *
01242                  *
01243                  *           NEXT
01244                  *           ----
01245                  *
01246                  *           THIS ROUTINE FIRSTLY ADDS THE STEP VALUE FOUND
01247                  *           IN THE CORRESPONDING "FOR" STATEMENT TO THE
01248                  *           VARIABLE VALUE, THEN DETERMINES WHETHER OR NOT THE
01249                  *           ASSOCIATED LOOP SHOULD BE TERMINATED. IF THE LOOP
01250                  *           IS TO CONTINUE THEN EXECUTION IS SET UP TO
01251                  *           CONTINUE AT THE FIRST STATEMENT FOLLOWING THE
01252                  *           "FOR" STATEMENT.
01253                  *
01254                  * I/P :
01255                  * XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
01256                  *
01257                  * *****
01258 2ED5 FE 20B5 NEXT    LDX XTEMP1   RETRIEVE LINE PTR
01259 2ED8 BD 376D        JSR FNDSPC   MOVE PTR PAST KEYWORD
01260 2EDB BD 3746        JSR CLASS    CLASSIFY CHARACTER
01261 2EDE C1 02          CMP B #02    IF CHAR IS NOT A
01262 2EE0 27 03          BEQ NEXT2    VARIABLE NAME THEN
01263 2EE2 7E 319E NEXT1  JMP SYNERR   SYNTAX ERROR OTHERWISE:
01264 2EE5 A6 00 NEXT2   LDA A 0,X     READ AND STORE
01265 2EE7 B7 20B2       STA A VARNAM  VARIABLE NAME
01266 2EEA 08           INX           MOVE LINE PTR PAST VARIABLE
01267 2EEB BD 37A3       JSR CHKCR2   CHECK FOR <CR> AT END LINE
01268 2EEE FE 20AF       LDX USRSP    SET UP TEMP USER SP
01269 2EF1 7F 208C       CLR FLAG     INDICATE NO GOSUBS ON STACK
01270 2EF4 8C 27FF NEXT3 CPX #USRSTK IF AT END OF STACK OR
01271 2EF7 27 15          BEQ NEXT5
01272 2EF9 A6 01          LDA A 1,X    IF CORRESPONDING "FOR" FOUND
01273 2EFB B1 20B2       CMP A VARNAM
01274 2EFE 27 14          BEQ NEXT6    THEN EXIT LOOP OTHERWISE:
01275 2F00 81 00          CMP A #00    IF GOSUB ADDR FOUND
01276 2F02 26 05          BNE NEXT4   THEN INDICATE GOSUB
01277 2F04 86 FF          LDA A #FF   ADDRESS EXISTS
01278 2F06 B7 208C       STA A FLAG   ON STACK
01279 2F09 08           NEXT4 INX
01280 2F0A 08           INX          DECREMENT TEMP STACK PTR
01281 2F0B 08           INX
01282 2F0C 20 E7          BRA NEXT3   REPEAT LOOP
01283 2F0E CE 4E46 NEXT5 LDX #NEXFOR  IF END OF STACK FOUND
01284 2F11 BD 31A1       JSR ERROR    THEN NEXT WITHOUT FOR ERROR
01285 2F14 7D 208C NEXT6 TST FLAG     IF GOSUB ADDR
01286 2F17 27 06          BEQ NEXT7   FOUND ON USER
01287 2F19 CE 4752       LDX #GOSRET  STACK THEN
01288 2F1C BD 31A1       JSR ERROR    GOSUB WITHOUT RETURN ERROR
01289 2F1F FF 20AF NEXT7 STX USRSP    STORE STACK PTR

```

```

01290 2F22 A6 02          LDA A 2,X
01291 2F24 B7 2096       STA A MSPART  RETRIEVE LINE NO.
01292 2F27 A6 03          LDA A 3,X
01293 2F29 B7 2095       STA A LSPART  OF "FOR" STATEMENT
01294 2F2C BD 3580       JSR FINDLN  FIND THAT LINE IN PROG
01295 2F2F 86 54         NEXT8 LDA A #'T
01296 2F31 A1 00          CMP A 0,X      SEARCH FOR
01297 2F33 26 06          BNE NEXT9
01298 2F35 86 4F          LDA A #'0      "TO" STATEMENT
01299 2F37 A1 01          CMP A 1,X
01300 2F39 27 03          BEQ NEXT10     IN THAT LINE
01301 2F3B 08             NEXT9 INX
01302 2F3C 20 F2          BRA NEXT8
01303 2F3E FF 20B5 NEXT10 STX XTEMP1    STORE LINE PTR
01304 2F41 BD 2C74       JSR TO         GET END LOOP AND STEP VALUES
01305 2F44 BD 35C9       JSR GETVAR     GET VARIABLE VALUE
01306 2F47 FF 20A2       STX RESULT     STORE IN RESULT
01307 2F4A BD 3423       JSR ADD        ADD STEP VALUE TO VAR VALUE
01308 2F4D BD 2EC1       JSR EXPRS5    AND REPLACE IN VARIABLE
01309 2F50 FE 20A2       LDX RESULT     GET NEW VAR VALUE
01310 2F53 7D 209C       TST OPRND1    IF STEP VALUE IS POSITIVE
01311 2F56 2B 0E         BMI NEXT12    AND VARIABLE VALUE
01312 2F58 8D 17         BSR DUDCPX    IS <= END LOOP VALUE
01313 2F5A 2F 0E         BLE NEXT13    THEN CONT EXECUTION OF LOOP
01314 2F5C FE 20AF NEXT11 LDX USRSP     END OF LOOP:
01315 2F5F 08            INX
01316 2F60 08            INX            DECREMENT STACK POINTER
01317 2F61 08            INX            (TO REMOVE LOOP VARIABLE)
01318 2F62 FF 20AF       STX USRSP
01319 2F65 39            RTS            IF STEP VALUE IS NEGATIVE
01320 2F66 8D 09         NEXT12 BSR DUDCPX AND VARIABLE VALUE
01321 2F68 2D F3         NEXT11 BLT NEXT11 IS >= END LOOP VALUE
01322 2F6A FE 20B5 NEXT13 LDX XTEMP1    THEN:
01323 2F6D 08            INX            INCREMENT LINE PTR PAST <CR>
01324 2F6E 7E 2D30       JMP ENDFR6    CONTINUE EXECUTION OF LOOP
01325 *
01326 *
01327 *          CPX EMULATE
01328 *          -----
01329 *
01330 *          THIS ROUTINE PERFORMS THE ASSEMBLER INSTRUCTION
01331 * "CPX ENDVAL". UNLIKE THE TRUE "CPX ENDVAL"
01332 * INSTRUCTION THIS ROUTINE SETS (AS APPROPRIATE) THE
01333 * Z, N AND V BITS IN THE CONDITION CODE REGISTER IN
01334 * THE WAY THAT M6800 CHIP SPECIFICATIONS SAY IT
01335 * SHOULD.
01336 *
01337 * I/P :
01338 * IX = INDEX REGISTER VALUE
01339 *
01340 * O/P :
01341 * Z, N AND V BITS SET APPROPRIATELY IN THE

```

```

01342          *   CONDITION CODE REGISTER
01343          *
01344          *****
01345 2F71 BC 2082 DUDCPX CPX      ENDVAL   SET THE Z BIT IF APPROPRIATE
01346 2F74 27 13      BEQ      DUDCP2   IF Z BIT SET THEN END
01347 2F76 FF 20AC     STX      TEMP     OTHERWISE STORE INDEX REGISTER
01348 2F79 B6 20AD     LDA A   TEMP+1   SUBTRACT ENDVAL
01349 2F7C B0 2083     SUB A   ENDVAL+1  FROM INDEX
01350 2F7F B6 20AC     LDA A   TEMP     REGISTER WHILE SETTING
01351 2F82 B2 2082     SBC A   ENDVAL   V AND N BITS AS
01352 2F85 07          DUDCP1 TPA       APPROPRIATE
01353 2F86 84 FB      AND A   $$FB     CLEAR Z BIT
01354 2F88 06          TAP
01355 2F89 39          DUDCP2 RTS       END OF "CPX ENDVAL"
01356          *
01357          *
01358          *           POKE
01359          *           ----
01360          *
01361          *   THIS ROUTINE STORES A SINGLE BYTE VALUE AT THE
01362          *   ADDRESS SPECIFIED IN THE "POKE" STATEMENT.
01363          *
01364          *   I/P :
01365          *   XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
01366          *
01367          *****
01368 2F8A FE 20B5 POKE  LDX      XTEMP1   RETRIEVE LINE PTR
01369 2F8D BD 376D     JSR      FNDSPC    MOVE PTR PAST KEYWORD
01370 2F90 BD 31E0     JSR      EVAL     EVALUATE EXPRESSION
01371 2F93 FE 20A2     LDX      RESULT   GET POKE ADDRESS
01372 2F96 FF 20BD     STX      XTEMP5   STORE POKE ADDRESS
01373 2F99 FE 20B5     LDX      XTEMP1   RETRIEVE LINE PTR
01374 2F9C 86 2C      LDA A   *,'      IF NEXT CHAR
01375 2F9E A1 00      CMP A   0,X       IS NOT A COMMA
01376 2FA0 27 03      BEQ      POKE1    THEN
01377 2FA2 7E 319E     JMP      SYNERR   SYNTAX ERROR OTHERWISE:
01378 2FA5 08          POKE1 INX        MOVE PTR PAST COMMA
01379 2FA6 BD 377A     JSR      SKPSPC    MOVE PTR TO NEXT NONSPACE
01380 2FA9 BD 31E0     JSR      EVAL     EVALUATE EXPRESSION
01381 2FAC FE 20B5     LDX      XTEMP1   RETRIEVE LINE PTR
01382 2FAF BD 37A3     JSR      CHKCR2   CHECK FOR <CR> AT END LINE
01383 2FB2 FE 20BD     LDX      XTEMP5   RETRIEVE POKE ADDRESS
01384 2FB5 B6 20A3     LDA A   RESULT+1  GET LS BYTE OF RESULT
01385 2FB8 A7 00      STA A   0,X       STORE RESULT AT POKE ADDR
01386 2FBA 39          RTS
01387          *
01388          *
01389          *           PRINT
01390          *           ----
01391          *
01392          *   THIS ROUTINE DISPLAYS ON THE SCREEN ALL ITEMS
01393          *   IN THE PRINT LIST FOLLOWING THE KEYWORD. THESE

```

```

01394          * ITEMS MAY BE STRINGS, NUMERIC RESULTS OF
01395          * EXPRESSIONS, AND/OR TAB POSITIONS.
01396          *
01397          * I/P :
01398          *   XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
01399          *
01400          *****
01401 2FBB FE 20B5 PRINT LDX   XTEMP1  RETRIEVE LINE POINTER
01402 2FBE BD 376D     JSR   FNDSPC  MOVE PTR PAST KEYWORD
01403 2FC1 7F 208F     CLR   LASTCH SET LAST CHAR NOT = ; OR ,
01404 2FC4 81 0D     PRINT1 CMP A  #CR  IF CURRENT CHAR IS A <CR>
01405 2FC6 26 03     BNE   PRINT2
01406 2FC8 7E 3061     JMP   PRINT5  THEN END OF PRINT
01407 2FCB 81 22     PRINT2 CMP A  #'  ELSE IF CURRENT CHAR IS A "
01408 2FCD 27 63     BEQ   STRING    THEN GO AND PRINT STRING
01409 2FCF 81 2C     CMP A  #' ,    ELSE IF CURRENT CHAR IS A ,
01410 2FD1 27 74     BEQ   TAB      THEN GO TO NEXT TAB POSITION
01411 2FD3 81 3B     CMP A  #' ;    ELSE IF CURRENT CHAR IS A ;
01412 2FD5 26 03     BNE   NUMB     THEN PROCESS DELIMITER
01413 2FD7 7E 3057     JMP   NOSPC   ELSE :
01414 2FDA BD 31E0 NUMB JSR   EVAL   EVALUATE EXPRESSION
01415 2FDD BD 3644     JSR   HEXASC  CONVERT RESULT TO ASCII
01416 2FE0 7D 20A6     TST   SIGN    IF RESULT IS NEGATIVE
01417 2FE3 27 07     BEQ   NUMB1
01418 2FE5 86 2D     LDA A  #' -
01419 2FE7 BD 70E7     JSR   OUT      THEN DISPLAY NEGATIVE SIGN
01420 2FEA 20 03     BRA   NUMB2
01421 2FEC BD 70C5 NUMB1 JSR   OUTS    OTHERWISE DISPLAY A SPACE
01422 2FEF BD 3078 NUMB2 JSR   UPDTAB  UPDATE TAB POSITION
01423 2FF2 86 30     LDA A  #' 0
01424 2FF4 CE 2097     LDX   #NUMBER  SET PTR AT START OF NO.
01425 2FF7 C6 04     LDA B  #$04
01426 2FF9 A1 00 NUMB3 CMP A  0,X    SKIP ANY
01427 2FFB 26 04     BNE   NUMB4
01428 2FFD 08       INX
01429 2FFE 5A       DEC B    LEADING ZEROS
01430 2FFF 26 F9     BNE   NUMB3
01431 3001 5C       NUMB4 INC B
01432 3002 A6 00 NUMB5 LDA A  0,X
01433 3004 BD 70E7     JSR   OUT      PRINT ALL REMAINING
01434 3007 F7 207F     STA B  COUNT
01435 300A 8D 6C     BSR   UPDTAB
01436 300C F6 207F     LDA B  COUNT
01437 300F 08       INX
01438 3010 5A       DEC B    DIGITS WHILE UPDATING
01439 3011 26 F0     BNE   NUMB5  THE TAB POSITION
01440 3013 BD 70C5     JSR   OUTS    PRINT TRAILING SPACE
01441 3016 8D 60     BSR   UPDTAB  UPDATE TAB POSITION
01442 3018 FE 20B5     LDX   XTEMP1  RETRIEVE LINE POINTER
01443 301B BD 377A PRINT3 JSR   SKPSPC  MOVE PTR TO NEXT NONSPACE
01444 301E 7F 208F     CLR   LASTCH SET LAST CHAR NOT = ; OR ,
01445 3021 A6 00     LDA A  0,X    READ NEXT CHARACTER

```

```

01446 3023 81 0D      CMP A  #CR      IF NEXT CHARACTER
01447 3025 27 9E      BEQ      PRINT1
01448 3027 81 3B      CMP A  #' ;      IS A <CR> OR A " ; "
01449 3029 27 9A      BEQ      PRINT1
01450 302B 81 2C      CMP A  #' ,      OR A " , " THEN NEXT PRINT ITEM
01451 302D 27 96      BEQ      PRINT1
01452 302F 7E 319E    PRINT4 JMP      SYNERR    DISPLAY SYNTAX ERROR MESSAGE
01453 3032 08          STRING INX          MOVE LINE PTR TO NEXT CHAR
01454 3033 A6 00      LDA A  0,X      IF A <CR> FOUND
01455 3035 81 0D      CMP A  #CR      BEFORE CLOSING QUOTES
01456 3037 27 F7      BEQ      PRINT4    THEN SYNTAX ERROR
01457 3039 81 22      CMP A  #' "      IF NEXT CHAR IS " THEN
01458 303B 26 03      BNE      STRNG1    END OF STRING SO
01459 303D 08          INX          MOVE LINE PTR PAST QUOTE
01460 303E 20 DC      BRA      PRINT3    AND DO END STRING PROCESSING
01461 3040 BD 70E7    STRNG1 JSR      OUT      OTHERWISE PRINT CURRENT CHAR
01462 3043 8D 33      BSR      UPDTAB    UPDATE TAB POSITION
01463 3045 20 EC      BRA      STRING    REPEAT FOR NEXT CHAR
01464 3047 C6 09      TAB      LDA B  #TABSIZ+1  CALCULATE NO. OF SPACES
01465 3049 F0 20A9    SUB B  TABPOS    UNTIL NEXT PRINT ZONE
01466 304C BD 73CB    JSR      NXTSPC    PRINT THAT MANY SPACES
01467 304F 7F 20A9    CLR      TABPOS    SET TAB POSITION
01468 3052 7C 20A9    INC      TABPOS    TO 1 AGAIN
01469 3055 86 2C      LDA A  #' ,      READ CHARACTER AGAIN
01470 3057 B7 208F    NOSPC  STA A  LASTCH    STORE LAST CHARACTER
01471 305A 08          INX          MOVE LINE PTR TO NEXT CHAR
01472 305B BD 377A    JSR      SKPSPC    MOVE LINE PTR TO NEXT NONSPACE
01473 305E 7E 2FC4    JMP      PRINT1    GO PROCESS NEXT PRINT ITEM
01474 3061 86 3B      PRINT5 LDA A  #' ;      END OF PRINT :
01475 3063 B1 208F    CMP A  LASTCH    IF LAST CHARACTER
01476 3066 27 0F      BEQ      PRINT6
01477 3068 86 2C      LDA A  #' ,      WAS NOT A ;
01478 306A B1 208F    CMP A  LASTCH
01479 306D 27 08      BEQ      PRINT6    OR A , THEN
01480 306F BD 7402    JSR      PCRLF    PRINT A <CR> AND <LF>
01481 3072 86 01      LDA A  #S01      RESET TAB POSITION
01482 3074 B7 20A9    STA A  TABPOS    TO 1 AGAIN
01483 3077 39          PRINT6 RTS          RETURN
01484 *
01485 *
01486 *          UPDATE TAB POSITION
01487 *          -----
01488 *
01489 *          THIS ROUTINE INCREMENTS THE TAB POSITION AND IF
01490 * IT EXCEEDS THE TAB SIZE IT IS RESET TO 1 AGAIN.
01491 *
01492 *****
01493 3078 7C 20A9    UPDTAB INC      TABPOS    INCREMENT TAB POSITION
01494 307B C6 08      LDA B  #TABSIZ    IF TAB POSITION > TAB SIZE
01495 307D F1 20A9    CMP B  TABPOS
01496 3080 2C 05      BGE      UPDTB1    THEN RESET TAB
01497 3082 C6 01      LDA B  #S01

```



```

01498 3084 F7 20A9          STA B   TABPOS   POSITION TO 1 AGAIN
01499 3087 39              UPDTB1 RTS
01500                      *
01501                      *
01502                      *          READ
01503                      *          ----
01504                      *
01505                      *          THIS ROUTINE ASSIGNS A VALUE TO EACH VARIABLE
01506                      *          IN THE VARIABLE LIST (WHICH FOLLOWS THE READ
01507                      *          STATEMENT) FROM THE CURRENT POSITION OF THE DATA
01508                      *          POINTER AND INCREMENTS THE DATA POINTER ONCE FOR
01509                      *          EVERY VALUE ASSIGNED.
01510                      *
01511                      * I/P :
01512                      *   XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
01513                      *
01514                      *****
01515 3088 FE 20B5 READ      LDX      XTEMP1   RETRIEVE LINE POINTER
01516 308B BD 376D          JSR      FNDSPC   MOVE PTR PAST KEYWORD
01517 308E BD 3746 READ1   JSR      CLASS   IF CHAR IS NOT ALPHABETIC
01518 3091 C1 02           CMP      B   #$02
01519 3093 27 03           BEQ      READ3
01520 3095 7E 319E READ2   JMP      SYNERR   DISPLAY SYNTAX ERROR MESSAGE
01521 3098 A6 00 READ3    LDA      A   0,X   OTHERWISE :
01522 309A B7 20B1          STA      A   VAR   STORE VARIABLE NAME
01523 309D 08              INX
01524 309E BD 377A          JSR      SKPSPC   SKIP TO NEXT NONSPACE
01525 30A1 81 0D           CMP      A   #CR   IF NEXT NONSPACE IS NOT
01526 30A3 27 08           BEQ      READ4   A <CR> OR A ",",
01527 30A5 81 2C           CMP      A   #'
01528 30A7 26 ED           BNE      READ2   THEN SYNTAX ERROR OTHERWISE
01529 30A9 08              INX
01530 30AA BD 377A          JSR      SKPSPC   MOVE PTR PAST COMMA AND
01531 30AD FF 20B5 READ4   STX      XTEMP1   SKIP ANY SPACES (COMMA ONLY)
01532 30B0 FE 2080          LDX      DATPTR  STORE LINE POINTER
01533 30B3 A6 00           LDA      A   0,X   RETRIEVE DATA POINTER
01534 30B5 81 0D           CMP      A   #CR
01535 30B7 26 2F           BNE      READ9   IF DATA POINTER IS
01536 30B9 08              INX               POINTING TO A <CR> THEN
01537 30BA BC 20A0 READ5   CPX      PROGEN  MOVE DATA PTR TO NEXT LINE
01538 30BD 26 06 READ6    BNE      READ7   IF DATA PTR AT
01539 30BF CE 4F44          LDX      #OUTDAT  END OF PROGRAM THEN
01540 30C2 BD 31A1          JSR      ERROR   OUT OF DATA SO
01541 30C5 A6 03 READ7    LDA      A   3,X   DISPLAY ERROR MESSAGE
01542 30C7 81 44           CMP      A   #'D   IF THE CURRENT LINE
01543 30C9 26 16           BNE      READ8
01544 30CB A6 04          LDA      A   4,X   BEGINS WITH A
01545 30CD 81 41           CMP      A   #'A
01546 30CF 26 10           BNE      READ8
01547 30D1 A6 05          LDA      A   5,X   DATA STATEMENT THEN
01548 30D3 81 54           CMP      A   #'T
01549 30D5 26 0A          BNE      READ8

```

```

01550 30D7 C6 06          LDA B  #06        MOVE DATA PTR TO
01551 30D9 BD 375D        JSR  IXADD       FIRST CHAR AFTER KEYWORD
01552 30DC BD 376D        JSR  FNDSPC      MOVE PTR PAST KEYWORD
01553 30DF 20 0B          BRA  READ10
01554 30E1 E6 02          READ8 LDA B  2,X    OTHERWISE :
01555 30E3 BD 375D        JSR  IXADD       MOVE DATA PTR TO NEXT LINE
01556 30E6 20 D3          BRA  READ6        AND REPEAT
01557 30E8 08            READ9 INX          IF DATA PTR WAS POINTING TO A
01558 30E9 BD 377A        JSR  SKPSPC      COMMA THEN SKIP COMMA
01559 30EC 7F 208C        READ10 CLR        ASCHEX NOT CALLED FROM INPUT
01560 30EF BD 36B2        JSR  ASCHEX     READ AND STORE DATA
01561 30F2 BD 377A        JSR  SKPSPC      MOVE PTR TO NEXT NONSPACE
01562 30F5 81 0D          CMP A  #CR       IF NEXT NONSPACE IS NOT
01563 30F7 27 04          BEQ  READ11      A <CR> OR A " , "
01564 30F9 81 2C          CMP A  #',
01565 30FB 26 99          BNE  READ2        THEN SYNTAX ERROR
01566 30FD FF 2080        READ11 STX        STORE DATA POINTER
01567 3100 BD 2EC7        JSR  PUTVAL     ASSIGN DATA TO VARIABLE
01568 3103 FE 20B5        LDX  XTEMP1     RETRIEVE LINE POINTER
01569 3106 86 0D          LDA A  #CR       IF LINE PTR CHAR IS NOT <CR>
01570 3108 A1 00          CMP A  0,X      THEN
01571 310A 27 03          BEQ  READ12      REPEAT ENTIRE PROCESS FOR
01572 310C 7E 308E        JMP  READ1      NEXT VARIABLE IN READ LIST
01573 310F 39            READ12 RTS        OTHERWISE FINISHED
01574 *
01575 *
01576 *          REM
01577 *          ---
01578 *
01579 *          THIS ROUTINE IGNORES THE REST OF THE PROGRAM
01580 *          LINE.
01581 *
01582 *          *****
01583 3110 39            REM  RTS          DUMMY "DO NOTHING" ROUTINE
01584 *
01585 *
01586 *          RETURN
01587 *          -----
01588 *
01589 *          THIS ROUTINE TRANSFERS CONTROL BACK TO THE LINE
01590 *          WHOSE LINE NUMBER IS THE FIRST FOUND ON THE STACK
01591 *          CORRESPONDING TO A GOSUB STACK ELEMENT. ANY LOOPS
01592 *          LEFT OPEN ARE DISCARDED BY REMOVING THEM FROM THE
01593 *          STACK WHILE SEARCHING FOR THE GOSUB STACK ELEMENT.
01594 *
01595 *          I/F :
01596 *          XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
01597 *
01598 *          *****
01599 3111 FE 20B5        RETURN LDX  XTEMP1     RETRIEVE LINE POINTER
01600 3114 BD 376D        JSR  FNDSPC      FIND SPACE DELIMITER
01601 3117 BD 37A3        JSR  CHKCR2     CHECK FOR <CR> AT END OF LINE

```

```

01602 311A FE 20AF      LDX      USRSP      RETRIEVE USER STACK PTR
01603 311D 8C 27FF RETUR1 CPX      #USRSTK   IF AT BOTTOM OF STACK
01604 3120 26 10      BNE      RETUR4      THEN :
01605 3122 FE 2091      LDX      LINENO     IF IN IMMEDIATE THEN
01606 3125 26 05      BNE      RETUR2      CAN'T CONTINUE ERROR
01607 3127 CE 4343      LDX      #NOCONT    IF IN RUN MODE
01608 312A 20 03      BRA      RETUR3      THEN SET UP FOR RETURN
01609 312C CE 5247 RETUR2 LDX      #RETGOS   WITHOUT GOSUB ERROR
01610 312F 7E 31A1 RETUR3 JMP      ERROR    DISPLAY ERROR MESSAGE
01611 3132 4F          RETUR4 CLR A        OTHERWISE:
01612 3133 A1 01      CMP A      1,X        CHECK IF LAST STACK ELEMENT
01613 3135 27 05      BEQ      RETUR5      WAS A RETURN ADDRESS
01614 3137 08          INX              IF NOT, THEN MOVE
01615 3138 08          INX              STACK POINTER TO
01616 3139 08          INX              NEXT STACK ELEMENT
01617 313A 20 E2      BRA      RETUR1      AND REPEAT THE ABOVE
01618 313C A6 03 RETUR5 LDA A      3,X        OTHERWISE:
01619 313E B7 2095      STA A      LSPART    EXTRACT LINE NUMBER
01620 3141 A6 02      LDA A      2,X
01621 3143 B7 2096      STA A      MSPART    FROM LAST ELEMENT
01622 3146 08          INX              MOVE THE STACK
01623 3147 08          INX              POINTER TO THE
01624 3148 08          INX              NEXT STACK ELEMENT
01625 3149 81 FF      CMP A      #$FF      IF LINE EXTRACTED
01626 314B 26 03      BNE      RETUR6      IS FFFF THEN
01627 314D 7E 2BD5      JMP      END1      END USER PROGRAM
01628 3150 FF 20AF RETUR6 STX      USRSP      SET USER STACK PTR
01629 3153 7E 2D67      JMP      GOTO2     GOTO EXTRACTED LINE NO.
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641

```

BREAK/STOP PROCESS

```

01635
01636
01637
01638
01639
01640
01641
01642 3156 7F 207C BREAK CLR      BRKFLG   CLEAR <BREAK> FLAG
01643 3159 20 09      BRA      STOP1      SKIP KEYWORD CHECK FOR "STOP"
01644 315B FE 20B5 STOP  LDX      XTEMP1    RETRIEVE SAVED BUFFER PTR
01645 315E BD 376D      JSR      FNDSPC    FIND SPACE DELIMITER
01646 3161 BD 37A3      JSR      CHKCR2   CHECK FOR <CR> AFTER KEYWORD
01647 3164 CE 2091 STOP1 LDX      #LINENO   SET UP LINE NUMBER
01648 3167 BD 360F      JSR      BCDNUM    IN STOPPED MESSAGE
01649 316A CE 37D2      LDX      #MESC4
01650 316D BD 703F      JSR      PSTRNG    DISPLAY STOPPED
01651 3170 CE 2087      LDX      #ERRLIN
01652 3173 BD 703F      JSR      PSTRNG    MESSAGE
01653 3176 BD 7402      JSR      PCRLF

```

```

01654 3179 7D 2091      TST      LINENO    IF CURRENTLY IN
01655 317C 26 05        BNE      STOP2
01656 317E 7D 2092      TST      LINENO+1    RUN MODE THEN
01657 3181 27 18        BEQ      STOP3
01658 3183 BD 3784 STOP2 JSR      CHKSTK    ENSURE NO STACK OVERFLOW
01659 3186 FE 20AF      LDX      USRSP      THEN PUSH THE
01660 3189 B6 209F      LDA A    PRGCNT+1
01661 318C A7 00        STA A    0,X        NEXT LINE NUMBER
01662 318E 09          DEX
01663 318F B6 209E      LDA A    PRGCNT    ONTO THE USER'S STACK
01664 3192 A7 00        STA A    0,X
01665 3194 09          DEX                AND DECREMENT THE
01666 3195 6F 00        CLR      0,X
01667 3197 09          DEX                USER'S STACK POINTER
01668 3198 FF 20AF      STX      USRSP
01669 319B 7E 2BDB STOP3 JMP      END2      GO BACK TO INTPTRT PROCESS
01670                      *
01671                      *
01672                      *          SYNTAX ERROR
01673                      *          -----
01674                      *
01675                      *          THIS ROUTINE DISPLAYS THE SYNTAX ERROR MESSAGE
01676                      * THEN RETURNS TO THE INTERPRETER PROCESS (USING
01677                      * THE ERROR ROUTINE BELOW).
01678                      *
01679                      * *****
01680 319E CE 534E SYNERR LDX      #SYNTAX    SET UP ERROR CODE
01681                      *
01682                      *
01683                      *          ERROR MESSAGE
01684                      *          -----
01685                      *
01686                      *          THIS ROUTINE DISPLAYS AN APPROPRIATE ERROR
01687                      * MESSAGE GIVEN THE ERROR CODE THEN RETURNS TO THE
01688                      * INTERPRETER PROCESS.
01689                      *
01690                      * I/P :
01691                      * IX = ERROR CODE.
01692                      *
01693                      * *****
01694 31A1 FF 2084 ERROR STX      ERRCOD    SET UP ERROR CODE
01695 31A4 CE 2091      LDX      #LINENO    SET UP LINE NUMBER
01696 31A7 BD 360F      JSR      BCDNUM    IN ERROR MESSAGE
01697 31AA BD 7402      JSR      PCRLF     PRINT A <CR>,<LF>
01698 31AD CE 2084      LDX      #ERRCOD
01699 31B0 BD 703F      JSR      PSTRNG    DISPLAY ERROR CODE
01700 31B3 CE 37C7      LDX      #MSG3
01701 31B6 BD 703F      JSR      PSTRNG    DISPLAY " ERROR IN "
01702 31B9 CE 2087      LDX      #ERRLIN
01703 31BC BD 703F      JSR      PSTRNG    DISPLAY LINE NUMBER
01704 31BF BD 7402      JSR      PCRLF     PRINT A <CR> , <LF>
01705 31C2 7E 2BD5      JMP      END1      JUMP TO END ROUTINE

```

```

01706      *
01707      *
01708      *          "WHAT?" MESSAGE
01709      *          -----
01710      *
01711      *          THIS ROUTINE DISPLAYS THE "WHAT?" MESSAGE IN
01712      *          RESPONSE TO AN OPERATING SYSTEM MODE OR IMMEDIATE
01713      *          MODE ERROR, THEN RETURNS TO THE INTERPRETER
01714      *          PROCESS.
01715      *
01716      *          ****
01717 31C5 CE 76CA WHAT LDX #MSG6 DISPLAY "WHAT?" MESSAGE
01718 31C8 BD 703F JSR PSTRNG
01719 31CB BD 7402 JSR PCRLF
01720 31CE 7E 2BDB JMP END2 GO BACK TO INTPT PROCESS
01721      *
01722      *
01723      *          BAD INPUT MESSAGE
01724      *          -----
01725      *
01726      *          THIS ROUTINE DISPLAYS THE MESSAGE FOR BAD INPUT
01727      *          THEN RETURNS TO THE BEGINNING OF THE INPUT ROUTINE
01728      *          AND STARTS INPUT AGAIN.
01729      *
01730      *          ****
01731 31D1 CE 37E0 BADINP LDX #MSG5
01732 31D4 BD 703F JSR PSTRNG DISPLAY BAD INPUT MESSAGE
01733 31D7 FE 20B7 LDX XTEMP2 RETRIEVE INITIAL LINE PTR
01734 31DA FF 20B5 STX XTEMP1 AND STORE BACK AS LINE PTR
01735 31DD 7E 2DF2 JMP INPUT RESTART INPUT ROUTINE
01736      *
01737      *
01738      *          EVALUATE EXPRESSION
01739      *          -----
01740      *
01741      *          THIS ROUTINE EVALUATES THE RESULT OF ANY
01742      *          RELATIONAL OR ARITHMETIC EXPRESSION STARTING FROM
01743      *          THE POSITION POINTED TO BY IX AND ENDING AT THE
01744      *          FIRST CHARACTER WHICH IS NOT PART OF THE
01745      *          EXPRESSION BUT IS THOUGHT TO BE LEGAL.
01746      *
01747      *          I/P :
01748      *          IX = POINTER TO START OF EXPRESSION
01749      *
01750      *          O/P :
01751      *          XTEMP1 = PTR TO FIRST CHAR AFTER EXPRESSION
01752      *          RESULT = EVALUATED VALUE OF EXPRESSION
01753      *
01754      *          ****
01755 31E0 FF 20B5 EVAL STX XTEMP1 STORE LINE POINTER
01756 31E3 BD 3784 JSR CHKSTK ENSURE NO STACK OVERFLOW
01757 31E6 FE 20AF LDX USRSP PUSH AN EXPRESSION MARKER

```

```

01758 31E9 09          DEX
01759 31EA 09          DEX
01760 31EB 86 01      LDA A  #01
01761 31ED A7 00      STA A  0,X
01762 31EF 09          DEX
01763 31F0 FF 20AF    STX  USRSP  USER'S STACK POINTER
01764 31F3 8D 16      BSR  PRCEXP  PROCESS EXPRESSION
01765 31F5 FE 20AF    LDX  USRSP  RETRIEVE USER'S STACK PTR
01766 31F8 A6 01      LDA A  1,X  IF NEXT STACK ELEMENT
01767 31FA 81 01      CMP A  #01  IS NOT AN EXPRESSION MARKER
01768 31FC 27 06      BEQ  EVAL1  THEN
01769 31FE CE 4545    LDX  #EXPERR  ERROR IN EXPRESSION
01770 3201 7E 31A1    JMP  ERROR  OTHERWISE
01771 3204 08          EVAL1 INX
01772 3205 08          INX
01773 3206 08          INX
01774 3207 FF 20AF    STX  USRSP  FROM USER STACK AND
01775 320A 39          RTS      RETURN
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796 320B BD 3784    PRCEXP JSR  CHKSTK  ENSURE NO STACK OVERFLOW
01797 320E FE 20AF    LDX  USRSP  PUSH OPERAND = 0
01798 3211 6F 00      CLR  0,X
01799 3213 09          DEX
01800 3214 6F 00      CLR  0,X
01801 3216 09          DEX
01802 3217 86 2B      LDA A  #' +
01803 3219 A7 00      STA A  0,X  AND INCREMENT USER'S
01804 321B 09          DEX
01805 321C FF 20AF    STX  USRSP  STACK POINTER
01806 321F FE 20B5    LDX  XTEMP1  RETRIEVE LINE POINTER
01807 3222 BD 3270    PRCEX1 JSR  PRUNOP  PROCESS UNARY OPERATOR
01808 3225 FF 20B5    STX  XTEMP1  STORE LINE POINTER
01809 3228 FE 20AF    LDX  USRSP  RETRIEVE STACK POINTER

```

```

01810 322B 09          DEX          INCREMENT USER
01811 322C 09          DEX          STACK POINTER
01812 322D 09          DEX          AND
01813 322E B6 20AE     LDA A UNOPTR  ADD UNARY OPERATOR
01814 3231 A7 01       STA A 1,X    TO USER STACK
01815 3233 FF 20AF     STX USRSP    STORE STACK POINTER
01816 3236 BD 32B0     JSR PROPND   PROCESS OPERAND
01817 3239 BD 3308     JSR PFUNOP   PERFORM UNARY OPERATION
01818 323C BD 3335     JSR PFB1OP   PERFORM BINARY OPERATION
01819 323F A6 00       LDA A 0,X    IF NOT END OF EXPRSN I.E.:
01820 3241 E6 01       LDA B 1,X
01821 3243 81 0D       CMP A #CR    <CR> NOT FOUND
01822 3245 27 25       BEQ PRCEX4
01823 3247 81 2C       CMP A #',    AND COMMA NOT FOUND
01824 3249 27 21       BEQ PRCEX4
01825 324B 81 3B       CMP A #';    AND SEMICOLON NOT FOUND
01826 324D 27 1D       BEQ PRCEX4
01827 324F 81 29       CMP A #')    AND END BRACKET NOT FOUND
01828 3251 27 19       BEQ PRCEX4
01829 3253 81 54       CMP A #'T    AND "TO" NOT FOUND
01830 3255 26 08       BNE PRCEX2
01831 3257 C1 4F       CMP B #'O
01832 3259 27 11       BEQ PRCEX4
01833 325B C1 48       CMP B #'H    AND "TH" (THEN) NOT FOUND
01834 325D 27 0D       BEQ PRCEX4
01835 325F 81 53       PRCEX2 CMP A #'S  AND "ST" (STEP) NOT FOUND
01836 3261 26 04       BNE PRCEX3
01837 3263 C1 54       CMP B #'T
01838 3265 27 05       BEQ PRCEX4   THEN :
01839 3267 BD 3392     PRCEX3 JSR PRB1OP  PROCESS BINARY OPERATOR AND
01840 326A 20 B7       BRA PRCEX1     REPEAT OTHERWISE :
01841 326C FF 20B5     PRCEX4 STX XTEMP1 STORE LINE POINTER AND
01842 326F 39          RTS          RETURN
01843                *
01844                *
01845                *      PROCESS UNARY OPERATOR
01846                *      -----
01847                *
01848                *      THIS ROUTINE DETERMINES WHAT UNARY OPERATOR
01849                *      EXISTS AT THE CURRENT LINE POINTER POSITION (IF
01850                *      ANY) AND SETS UP A UNARY OPERATOR FLAG
01851                *      ACCORDINGLY. THE LINE POINTER IS THEN MOVED TO
01852                *      THE OPERAND IF NECESSARY.
01853                *
01854                *      I/P :
01855                *      IX = LINE POINTER POSITION
01856                *
01857                *      O/P :
01858                *      IX = LINE PTR POSITION POINTING TO NEXT OPERAND
01859                *      UNOPTR = UNARY OPERATOR FLAG
01860                *      = 2B (+) FOR UNARY +
01861                *      = 2D (-) FOR UNARY -

```

```

01862          *          = 27 (') FOR UNARY NOT
01863          *
01864          *****
01865 3270 86 2B PRUNOP LDA A #' + ASSUME UNARY OPERATOR
01866 3272 B7 20AE STA A UNOPTR IS UNARY +
01867 3275 A6 00 LDA A 0,X
01868 3277 81 2B CMP A #' + IF CHAR IS "+" THEN
01869 3279 27 20 BEQ PRUN03 PROCESS UNARY +
01870 327B 81 2D CMP A #' - IF CHAR IS "-" THEN
01871 327D 27 1C BEQ PRUN03 PROCESS UNARY -
01872 327F 81 4E CMP A #' N IF NEXT 3 CHARS
01873 3281 26 2C BNE PRUN04
01874 3283 A6 01 LDA A 1,X
01875 3285 81 4F CMP A #' 0 ARE "NOT" THEN
01876 3287 26 26 BNE PRUN04
01877 3289 A6 02 LDA A 2,X
01878 328B 81 54 CMP A #' T
01879 328D 26 20 BNE PRUN04
01880 328F 08 PRUN01 INX INCREMENT LINE PTR
01881 3290 08 INX TWICE
01882 3291 86 27 LDA A #' GET OPERATOR FLAG FOR "NOT"
01883 3293 08 PRUN02 INX MOVE LINE PTR PAST OPERATOR
01884 3294 B7 20AE STA A UNOPTR STORE UNARY OPERATOR FLAG
01885 3297 BD 377A JSR SKPSPC MOVE PTR TO NEXT NONSPACE
01886 329A 39 RTS
01887 329B FF 20B5 PRUN03 STX XTEMP1 STORE LINE POINTER
01888 329E 8D F4 BSR PRUN02 STORE UNARY OPERATOR FLAG
01889 32A0 BD 3746 JSR CLASS IF NEXT OPERAND IS
01890 32A3 C1 01 CMP B #$01 NUMERIC THEN
01891 32A5 26 08 BNE PRUN04
01892 32A7 FE 20B5 LDX XTEMP1 NO UNARY OPERATOR
01893 32AA 86 2B LDA A #' + (WILL FIX WHEN READ NO.)
01894 32AC B7 20AE STA A UNOPTR OTHERWISE
01895 32AF 39 PRUN04 RTS LEAVE UNARY OPERATOR AS IS
01896          *
01897          *
01898          * PROCESS OPERAND
01899          * -----
01900          *
01901          * THIS ROUTINE OBTAINS THE VALUE FOR THE OPERAND
01902          * AT THE LINE POINTER, WHERE THE OPERAND MAY BE THE
01903          * VALUE OBTAINED FROM A FUNCTION, A VARIABLE, A
01904          * CONSTANT, OR ANOTHER EXPRESSION ENCLOSED IN
01905          * BRACKETS.
01906          *
01907          * I/P :
01908          * XTEMP1 = POINTER TO OPERAND
01909          *
01910          * O/P :
01911          * IX = LINE PTR POSITION POINTING TO THE FIRST
01912          * NONSPACE CHAR AFTER THE OPERAND
01913          * RESULT = VALUE OF OPERAND

```



```

01914      *
01915      *****
01916 32B0 CE 2ABA PRPNDD LDX #FNTBL SET UP TABLE POINTER AT
01917 32B3 FF 20AA STX TBLPTR START OF FUNCTION TABLE
01918 32B6 FE 20B5 LDX XTEMP1 RETRIEVE LINE POINTER
01919 32B9 BD 2A00 JSR FINDKE LOOK FOR FUNCTION IN
01920 32BC 5D TST B FUNCTION KEYWORD TABLE
01921 32BD 27 0A BEQ PRVAR IF KEYWORD FOUND THEN :
01922 32BF AD 00 PRFN JSR 0,X PROCESS FUNCTION
01923 32C1 FE 20B5 LDX XTEMP1 RETRIEVE LINE PTR
01924 32C4 08 PRFN1 INX MOVE PTR PAST END BRACKET
01925 32C5 BD 377A PRFN2 JSR SKPSPC MOVE PTR TO NEXT NONSPACE
01926 32C8 39 RTS OTHERWISE :
01927 32C9 FE 20B5 PRVAR LDX XTEMP1
01928 32CC BD 3746 JSR CLASS IF CHAR AT PTR IS
01929 32CF C1 02 CMP B #02 ALPHABETIC THEN :
01930 32D1 26 11 BNE PRBRKT
01931 32D3 A6 00 LDA A 0,X READ VARIABLE NAME
01932 32D5 B7 20B1 STA A VAR
01933 32D8 BD 35CF JSR GETVAL RETRIEVE VARIABLE VALUE
01934 32DB FF 20A2 STX RESULT STORE IN RESULT
01935 32DE FE 20B5 LDX XTEMP1 RETRIEVE LINE POINTER
01936 32E1 08 INX MOVE PTR PAST VARIABLE NAME
01937 32E2 20 E2 BRA PRFN2 OTHERWISE :
01938 32E4 A6 00 PRBRKT LDA A 0,X IF CHAR AT PTR IS
01939 32E6 81 28 CMP A #'( AN OPENING BRACKET THEN :
01940 32E8 26 19 BNE PRCON
01941 32EA 08 INX MOVE LINE PTR PAST BRACKET
01942 32EB BD 377A JSR SKPSPC MOVE PTR TO NEXT NONSPACE
01943 32EE FF 20B5 STX XTEMP1 STORE LINE PTR
01944 32F1 BD 320B JSR PRCEXP PROCESS EXPRSN (RECURSION!)
01945 32F4 FE 20B5 LDX XTEMP1 RETRIEVE LINE POINTER
01946 32F7 A6 00 LDA A 0,X
01947 32F9 81 29 CMP A #' ) ENSURE NEXT CHAR IS A
01948 32FB 27 C8 BEQ PRFN1 CLOSING BRACKET - IF NOT :
01949 32FD CE 4545 LDX #EXPERR ERROR IN EXPRESSION
01950 3300 7E 31A1 JMP ERROR OTHERWISE (NOT OPEN BRACKET):
01951 3303 BD 36B2 PRCON JSR ASCHEX MUST BE A CONSTANT SO
01952 3306 20 BE BRA PRFN2 PROCESS CONSTANT AND RETURN
01953      *
01954      *
01955      *
01956      *
01957      *
01958      * THIS ROUTINE PERFORMS THE UNARY OPERATION
01959      * DEFINED IN UNOPTR ON THE OPERAND FOUND IN RESULT.
01960      *
01961      * I/P :
01962      * RESULT = OPERAND FOR UNARY OPERATION
01963      *
01964      * O/P :
01965      * OPRND1 = ANSWER AFTER UNARY OPERATION

```

```

01966      *
01967      *****
01968 3308 FF 20B5 PFUNOP STX  XTEMP1  STORE LINE POINTER
01969 330B FE 20AF      LDX  USRSP    GET STACK POINTER
01970 330E A6 01      LDA  A  1,X    GET UNARY OPERATOR
01971 3310 08      INX                FROM USER STACK
01972 3311 08      INX                AND DECREMENT USER
01973 3312 08      INX                STACK POINTER
01974 3313 FF 20AF      STX  USRSP    STORE STACK POINTER
01975 3316 81 2B      CMP  A  #' +   IF UNARY OPERATOR = + THEN
01976 3318 26 0A      BNE  PFNEG
01977 331A FE 20A2 PFUN01 LDX  RESULT  GET RESULT AND
01978 331D FF 209C      STX  OPRND1   STORE IN OPRND1
01979 3320 FE 20B5      LDX  XTEMP1   RETRIEVE LINE PTR AND
01980 3323 39      RTS                RETURN
01981 3324 81 2D      PFNEG CMP  A  #' -   IF UNARY OPERATOR = - THEN
01982 3326 26 08      BNE  PFNOT
01983 3328 CE 20A2      LDX  #RESULT
01984 332B BD 3405      JSR  NEGATE    NEGATE OPERAND AND
01985 332E 20 EB      BRA  PFUN01     RETURN
01986 3330 BD 3516 PFNOT JSR  NOT      IF UNARY OP = NOT THEN
01987 3333 20 E6      BRA  PFUN01     PERFORM NOT AND RETURN
01988      *
01989      *
01990      *          PERFORM BINARY OPERATION
01991      *          -----
01992      *
01993      *          THIS ROUTINE PERFORMS THE BINARY OPERATION
01994      *          FOUND IN THE LAST ELEMENT OF THE USERS STACK ON
01995      *          TWO OPERANDS, ONE OF WHICH WAS ALSO IN THE LAST
01996      *          ELEMENT OF THE USERS STACK, AND THE OTHER IS
01997      *          CONTAINED IN RESULT.
01998      *
01999      * I/P :
02000      *          OPRND1 = ONE OPERAND FOR BINARY OPERATOR
02001      *
02002      * O/P :
02003      *          RESULT = ANSWER AFTER BINARY OPERATION
02004      *
02005      *          *****
02006 3335 FF 20B5 PFBIOF STX  XTEMP1  STORE LINE POINTER
02007 3338 FE 20AF      LDX  USRSP    RETRIEVE STACK POINTER
02008 333B EE 02      LDX  2,X      GET SECOND OPERAND AND
02009 333D FF 20A2      STX  RESULT   STORE IN RESULT
02010 3340 FE 20AF      LDX  USRSP
02011 3343 E6 01      LDA  B  1,X    GET BINARY OPERATOR
02012 3345 08      INX
02013 3346 08      INX                DECREMENT STACK POINTER
02014 3347 08      INX
02015 3348 FF 20AF      STX  USRSP    STORE STACK POINTER
02016 334B FE 20B5      LDX  XTEMP1   RETRIEVE LINE POINTER
02017 334E C1 2B      CMP  B  #' +   IF BINARY OPERATOR = +

```

```

02018 3350 26 04      BNE      PFSUBT  THEN
02019 3352 BD 3423 PFADD JSR      ADD      PERFORM ADDITION
02020 3355 39          RTS          OTHERWISE
02021 3356 C1 2D      PFSUBT CMP B  #' -   IF BINARY OPERATOR = -
02022 3358 26 04      BNE      PFMULT  THEN
02023 335A BD 343E      JSR      SUBT     PERFORM SUBTRACTION
02024 335D 39          RTS          OTHERWISE
02025 335E C1 2A      PFMULT CMP B  #' *   IF BINARY OPERATOR = *
02026 3360 26 04      BNE      PFDIV   THEN
02027 3362 BD 3451      JSR      MULT    PERFORM MULTIPLICATION
02028 3365 39          RTS          OTHERWISE
02029 3366 C1 2F      PFDIV  CMP B  #' /   IF BINARY OPERATOR = /
02030 3368 26 04      BNE      PFAND    THEN
02031 336A BD 3494      JSR      DIVIDE  PERFORM INTEGER DIVISION
02032 336D 39          RTS          OTHERWISE
02033 336E C1 26      PFAND  CMP B  #' &   IF BINARY OPERATOR = AND
02034 3370 26 04      BNE      PFOR     THEN
02035 3372 BD 34FA      JSR      AND     PERFORM LOGICAL AND
02036 3375 39          RTS          OTHERWISE
02037 3376 C1 23      PFOR   CMP B  #' #   IF BINARY OPERATOR = OR
02038 3378 26 04      BNE      PFEQL    THEN
02039 337A BD 3508      JSR      OR      PERFORM LOGICAL OR
02040 337D 39          RTS          OTHERWISE
02041 337E C1 3D      PFEQL  CMP B  #' =   IF BINARY OPERATOR = "="
02042 3380 26 04      BNE      PFLT     THEN
02043 3382 BD 351F      JSR      EQUALS  PERFORM EQUALS RELATION
02044 3385 39          RTS          OTHERWISE
02045 3386 C1 3C      PFLT   CMP B  #' <   IF BINARY OPERATOR = <
02046 3388 26 04      BNE      PFGT     THEN
02047 338A BD 352A      JSR      LT      PERFORM LESS THAN RELATION
02048 338D 39          RTS          OTHERWISE
02049 338E BD 3530 PFGT JSR      GT      PERFORM GREATER THAN RELN
02050 3391 39          RTS
02051          *
02052          *
02053          *      PROCESS BINARY OPERATOR
02054          *      -----
02055          *
02056          *      THIS ROUTINE DETERMINES WHAT BINARY OPERATOR
02057          *      EXISTS AT THE CURRENT LINE POINTER POSITION AND
02058          *      SETS UP THIS BINARY OPERATOR IN THE USERS STACK
02059          *      ALONG WITH AN OPERAND FOR IT: THE RESULT OF THE
02060          *      EXPRESSION UP TO THE CURRENT POSITION OF THE LINE
02061          *      POINTER. THE LINE POINTER IS MOVED PAST THE
02062          *      OPERATOR TO THE NEXT OPERAND.
02063          *
02064          *      I/P :
02065          *      IX = POINTER TO BINARY OPERATOR
02066          *      RESULT = RESULT OF EXPRESSION UP TO CURRENT
02067          *      LINE POINTER POSITION
02068          *
02069          *      O/P :

```

```

02070          *      IX = POINTER TO OPERAND FOLLOWING THE BINARY
02071          *      OPERATOR
02072          *
02073          *****
02074 3392 FF 20B5 PRBIO1P STX      XTEMP1    STORE LINE PTR
02075 3395 BD 3784          JSR      CHKSTK    ENSURE NO STACK OVERFLOW
02076 3398 FE 20AF          LDX      USRSP    RETRIEVE STACK POINTER
02077 339B B6 20A3          LDA A    RESULT+1  GET LS BYTE OF RESULT
02078 339E A7 00          STA A    0,X      AND STORE IN USER STACK
02079 33A0 09          DEX
02080 33A1 B6 20A2          LDA A    RESULT    GET MS BYTE OF RESULT
02081 33A4 A7 00          STA A    0,X      AND STORE IN USER STACK
02082 33A6 09          DEX
02083 33A7 09          DEX
02084 33AB FF 20AF          STX      USRSP    INCREMENT STACK POINTER
02085 33AB FE 20B5          LDX      XTEMP1    STORE STACK POINTER
02086 33AE A6 00          LDA A    0,X      RETRIEVE LINE POINTER
02087 33B0 81 2B          CMP A    #'+'    GET BINARY OPERATOR CHAR
02088 33B2 27 23          BEQ      PRBIO1    IF CHAR IS "+" THEN
02089 33B4 81 2D          CMP A    #'-'    STORE BINARY OPERATOR
02090 33B6 27 1F          BEQ      PRBIO1    IF CHAR IS "-" THEN
02091 33B8 81 2A          CMP A    #'*'    STORE BINARY OPERATOR
02092 33BA 27 1B          BEQ      PRBIO1    IF CHAR IS "*" THEN
02093 33BC 81 2F          CMP A    #'/'    STORE BINARY OPERATOR
02094 33BE 27 17          BEQ      PRBIO1    IF CHAR IS "/" THEN
02095 33C0 81 41          CMP A    #'A'    STORE BINARY OPERATOR
02096 33C2 26 20          BNE      PRBIO2    IF NEXT 3 CHARS
02097 33C4 A6 01          LDA A    1,X
02098 33C6 81 4E          CMP A    #'N'
02099 33C8 26 38          BNE      PRBIO4    ARE "AND" THEN :
02100 33CA A6 02          LDA A    2,X
02101 33CC 81 44          CMP A    #'D'
02102 33CE 26 32          BNE      PRBIO4
02103 33D0 08          INX
02104 33D1 08          INX
02105 33D2 FF 20B5          STX      XTEMP1    INCREMENT LINE PTR TWICE
02106 33D5 86 26          LDA A    #'&'    STORE LINE POINTER
02107 33D7 FE 20AF PRBIO1P LDX      USRSP    GET OPERATOR FLAG FOR "AND"
02108 33DA A7 01          STA A    1,X      RETRIEVE USER STACK PTR
02109 33DC FE 20B5          LDX      XTEMP1    STORE OPERATOR IN USER STACK
02110 33DF 08          INX
02111 33E0 BD 377A          JSR      SKPSPC    RETRIEVE LINE POINTER
02112 33E3 39          RTS                MOVE LINE PTR PAST OPERATOR
02113 33E4 81 4F          PRBIO2 CMP A    #'O'    MOVE PTR TO NEXT NONSPACE
02114 33E6 26 0E          BNE      PRBIO3    AND RETURN
02115 33E8 A6 01          LDA A    1,X      IF NEXT 2 CHARS
02116 33EA 81 52          CMP A    #'R'
02117 33EC 26 14          BNE      PRBIO4    ARE "OR" THEN :
02118 33EE 08          INX
02119 33EF FF 20B5          STX      XTEMP1    INCREMENT LINE POINTER
02120 33F2 86 23          LDA A    #'#'    STORE LINE POINTER
02121 33F4 20 E2          BRA      PRBIO1    GET OPERATOR FLAG FOR "OR"
                                AND STORE BINARY OPERATOR

```

```

02122 33F6 81 3D   PRBI03 CMP A #'=      IF CHAR IS "=" THEN
02123 33F8 27 DE           BEQ   PRBI01  STORE BINARY OPERATOR
02124 33FA 81 3C           CMP A #'<    IF CHAR IS "<" THEN
02125 33FC 27 DA           BEQ   PRBI01  STORE BINARY OPERATOR
02126 33FE 81 3E           CMP A #'>    IF CHAR IS ">" THEN
02127 3400 27 D6           BEQ   PRBI01  STORE BINARY OPERATOR
02128 3402 7E 319E PRBI04 JMP   SYNERR  OTHERWISE SYNTAX ERROR
02129                *
02130                *
02131                *          NEGATE
02132                *          -----
02133                *
02134                *          THIS ROUTINE NEGATES A 2 BYTE, 2'S  COMPLEMENT
02135                *          HEXADECIMAL NUMBER.
02136                *          I.E. NUMB := - NUMB
02137                *
02138                *          I/P :
02139                *          IX = POINTER TO NUMBER TO BE NEGATED
02140                *
02141                *          *****
02142 3405 86 80   NEGATE LDA A #80      TEST IF NUMBER
02143 3407 A1 00           CMP A 0,X
02144 3409 26 0A           BNE   NEGAT1  TRYING TO NEGATE IS
02145 340B 6D 01           TST   1,X
02146 340D 26 06           BNE   NEGAT1  -32768 IF SO THEN
02147 340F CE 4F56        LDX   #ARITOV  SETUP FOR ARITH OVERFLOW AND
02148 3412 7E 31A1        JMP   ERROR    DISPLAY ERROR MESSAGE ELSE
02149 3415 63 00   NEGAT1 COM   0,X      COMPLEMENT ENTIRE
02150 3417 A6 01           LDA A 1,X      NUMBER THEN
02151 3419 43           COM A
02152 341A 88 01           ADD A #01      ADD 1 TO LOWER ORDER BYTE
02153 341C A7 01           STA A 1,X
02154 341E 24 02           BCC   NEGAT2  IF CARRY GENERATED THEN
02155 3420 6C 00           INC   0,X      ADD 1 TO HIGHER ORDER BYTE
02156 3422 39           NEGAT2 RTS
02157                *
02158                *
02159                *          ADD
02160                *          ---
02161                *
02162                *          THIS ROUTINE PERFORMS THE ADDITION OF TWO 2
02163                *          BYTE, 2'S COMPLEMENT HEXADECIMAL NUMBERS, NAMELY
02164                *          RESULT AND OPRND1, AND PUTS THE ANSWER IN RESULT.
02165                *          I.E. RESULT := RESULT + OPRND1
02166                *
02167                *          I/P :
02168                *          RESULT = NUMBER TO BE ADDED TO
02169                *          OPRND1 = NUMBER TO ADD
02170                *
02171                *          O/P :
02172                *          RESULT = RESULT OF ADDITION
02173                *

```

```

02174 *****
02175 3423 B6 20A3 ADD LDA A RESULT+1 ADD LOWER ORDER
02176 3426 B8 209D ADD A OPRND1+1 BYTES TOGETHER
02177 3429 B7 20A3 STA A RESULT+1 STORE BACK IN RESULT
02178 342C B6 20A2 LDA A RESULT ADD HIGHER ORDER BYTES
02179 342F B9 209C ADC A OPRND1 INCLUDING ANY CARRY FROM
02180 3432 28 06 BVC ADD1 PREVIOUS ADDITION
02181 3434 CE 4F56 LDX #ARITOV IF ARITHMETIC OVERFLOW THEN
02182 3437 7E 31A1 JMP ERROR DISPLAY ERROR MESSAGE
02183 343A B7 20A2 ADD1 STA A RESULT ELSE STORE BACK IN RESULT
02184 343D 39 RTS
02185 *
02186 *
02187 * SUBTRACT
02188 * -----
02189 *
02190 * THIS ROUTINE PERFORMS THE SUBTRACTION OF TWO
02191 * 2 BYTE, 2'S COMPLEMENT HEXADECIMAL NUMBERS, NAMELY
02192 * RESULT AND OPRND1, AND PUTS THE ANSWER IN RESULT.
02193 * I.E. RESULT := RESULT - OPRND1
02194 *
02195 * I/P :
02196 * RESULT = NUMBER TO BE SUBTRACTED FROM
02197 * OPRND1 = NUMBER TO SUBTRACT
02198 *
02199 * O/P :
02200 * RESULT = RESULT OF SUBTRACTION
02201 *
02202 *****
02203 343E FF 20B3 SUBT STX XTEMP0 SAVE IX
02204 3441 CE 209C LDX #OPRND1
02205 3444 8D C0 BSR NEGATE NEGATE OPERAND 1
02206 3446 8D DC BSR ADD ADD OPERAND 1 TO RESULT
02207 3448 CE 209C LDX #OPRND1
02208 344B 8D B9 BSR NEGATE NEGATE OPERAND 1 BACK AGAIN
02209 344D FE 20B3 LDX XTEMP0 RESTORE IX
02210 3450 39 RTS
02211 *
02212 *
02213 * MULTIPLY
02214 * -----
02215 *
02216 * THIS ROUTINE PERFORMS THE MULTIPLICATION OF TWO
02217 * 2 BYTE, 2'S COMPLEMENT HEXADECIMAL NUMBERS, NAMELY
02218 * RESULT AND OPRND1, AND PUTS THE ANSWER IN RESULT.
02219 * I.E. RESULT := RESULT * OPRND1
02220 *
02221 * I/P :
02222 * RESULT = NUMBER TO BE MULTIPLIED
02223 * OPRND1 = NUMBER TO MULTIPLY BY
02224 *
02225 * O/P

```

```

02226          *   RESULT = RESULT OF MULTIPLICATION
02227          *
02228          *****
02229 3451 FF 20B3 MULT STX  XTEMP0  SAVE IX
02230 3454 7F 20A6      CLR  SIGN    ASSUME OPERAND 1 IS POSITIVE
02231 3457 B6 209C      LDA A  OPRND1  IF OPERAND 1 IS NEGATIVE
02232 345A 84 80        AND A  #NEGMSK
02233 345C 27 08        BEQ  MULT1    THEN CHANGE SIGN TO
02234 345E 73 20A6      COM  SIGN
02235 3461 CE 209C      LDX  #OPRND1  INDICATE THIS AND
02236 3464 8D A0        BSR  NEGATE    MAKE OPERAND 1 POSITIVE
02237 3466 FE 209C MULT1 LDX  OPRND1  GET NO. OF REPEATED ADDS
02238 3469 B6 20A2      LDA A  RESULT  SET UP OPERAND 1
02239 346C B7 209C      STA A  OPRND1
02240 346F B6 20A3      LDA A  RESULT+1  EQUAL TO RESULT
02241 3472 B7 209D      STA A  OPRND1+1
02242 3475 7F 20A2      CLR  RESULT    INITIALISE RESULT
02243 3478 7F 20A3      CLR  RESULT+1  TO ZERO
02244 347B 8C 0000      CPX  #0000    IF NO REPEATED ADDITIONS
02245 347E 27 05        BEQ  MULT3    THEN FINISHED
02246 3480 8D A2  MULT2 BSR  ADD      OTHERWISE ADD ONCE
02247 3482 09          DEX            DECREMENT NO OF REPEATED
02248 3483 26 FC        BNE  MULT2    ADDS AND REPEAT
02249 3485 7D 20A6 MULT3 TST  SIGN    IF OPERAND 1 WAS ORIGINALLY
02250 3488 27 06        BEQ  MULT4    NEGATIVE THEN
02251 348A CE 20A2      LDX  #RESULT
02252 348D BD 3405      JSR  NEGATE    NEGATE RESULT
02253 3490 FE 20B3 MULT4 LDX  XTEMP0  RESTORE IX
02254 3493 39          RTS
02255          *
02256          *
02257          *   DIVIDE
02258          *   -----
02259          *
02260          *   THIS ROUTINE PERFORMS THE DIVISION OF TWO 2
02261          *   BYTE, 2'S COMPLEMENT HEXADECIMAL NUMBERS, NAMELY
02262          *   RESULT AND OPRND1, AND PUTS THE ANSWER IN RESULT.
02263          *   I.E. RESULT := RESULT DIV OPRND1
02264          *
02265          *   I/P :
02266          *   RESULT = DIVIDEND
02267          *   OPRND1 = DIVISOR
02268          *
02269          *   O/P :
02270          *   RESULT = RESULT OF DIVISION
02271          *   OPRND1 = REMAINDER FROM DIVISION
02272          *
02273          *****
02274 3494 FF 20B8 DIVIDE STX  XTEMP4  SAVE IX
02275 3497 FE 209C      LDX  OPRND1  IF DIVISOR IS ZERO THEN
02276 349A 26 06        BNE  DIVID1  DIVIDE BY ERROR SO SET
02277 349C CE 2F30      LDX  #DIVZER  UP ERROR CODE AND

```

```

02278 349F 7E 31A1      JMP      ERROR      DISPLAY ERROR MESSAGE
02279 34A2 7F 20A6 DIVID1 CLR      SIGN      ASSUME SIGN IS POSITIVE
02280 34A5 B6 20A2      LDA A    RESULT    IF SIGN OF RESULT OPERAND
02281 34A8 84 80        AND A    #NEGMSK   IS NEGATIVE THEN
02282 34AA 27 09        BEQ      DIVID2
02283 34AC 73 20A6      COM      SIGN      COMPLEMENT SIGN
02284 34AF CE 20A2      LDX      #RESULT
02285 34B2 BD 3405      JSR      NEGATE    AND NEGATE RESULT OPERAND
02286 34B5 B6 209C DIVID2 LDA A    OPRND1   IF SIGN OF OPERAND1
02287 34B8 84 80        AND A    #NEGMSK   IS NEGATIVE THEN
02288 34BA 27 09        BEQ      DIVID3
02289 34BC 73 20A6      COM      SIGN      COMPLEMENT SIGN
02290 34BF CE 209C      LDX      #OPRND1
02291 34C2 BD 3405      JSR      NEGATE
02292 34C5 CE FFFF DIVID3 LDX      #-1      INITIALISE ANSWER = -1
02293 34C8 BD 343E DIVID4 JSR      SUBT      SUBTRACT OPRND1 FROM RESULT
02294 34CB 08           INX              SET ANSWER UP BY 1
02295 34CC B6 20A2      LDA A    RESULT    IF RESULT OF SUBTRACTION IS
02296 34CF 84 80        AND A    #NEGMSK   STILL > ZERO THEN CONTINUE
02297 34D1 27 F6        BEQ      DIVID4    SUBTRACTING OTHERWISE FINISH
02298 34D3 BD 3423      JSR      ADD      CALCULATE REMAINDER
02299 34D6 B6 20A2      LDA A    RESULT    STORE REMAINDER
02300 34D9 B7 209C      STA A    OPRND1
02301 34DC B6 20A3      LDA A    RESULT+1   INTO OPERAND 1
02302 34DF B7 209D      STA A    OPRND1+1
02303 34E2 FF 20A2      STX      RESULT    STORE ANSWER IN RESULT
02304 34E5 7D 20A6      TST      SIGN      IF SIGN OF ANSWER
02305 34E8 27 0C        BEQ      DIVID5    IS NEGATIVE THEN
02306 34EA CE 20A2      LDX      #RESULT
02307 34ED BD 3405      JSR      NEGATE    NEGATE RESULT
02308 34F0 CE 209C      LDX      #OPRND1
02309 34F3 BD 3405      JSR      NEGATE    AND NEGATE REMAINDER
02310 34F6 FE 20BB DIVID5 LDX      XTEMP4   RESTORE IX
02311 34F9 39          RTS
02312                *
02313                *
02314                *      AND
02315                *      ---
02316                *
02317                *      THIS ROUTINE PERFORMS THE BOOLEAN 'AND' ON TWO
02318                *      NUMBERS, NAMELY RESULT AND OPRND1 ACCORDING TO THE
02319                *      FOLLOWING TABLE :
02320                *
02321                *      RESULT      | OPRND1      | RESULT AND OPRND1
02322                *      -----+-----+-----
02323                *      0000      | 0000        | 0000
02324                *      0000      | OTHERWISE    | 0000
02325                *      OTHERWISE | 0000        | 0000
02326                *      OTHERWISE | OTHERWISE    | 0001
02327                *
02328                *      I/P :
02329                *      RESULT = FIRST NUMBER

```



```

02330      *           = 0000 IF FALSE
02331      *           = OTHERWISE IF TRUE
02332      *   OPRND1 = SECOND NUMBER
02333      *           = 0000 IF FALSE
02334      *           = OTHERWISE IF TRUE
02335      *
02336      * O/P :
02337      *   RESULT = BOOLEAN ANSWER
02338      *           = 0000 IF FALSE
02339      *           = 0001 IF TRUE
02340      *
02341      * *****
02342 34FA 8D 3A   AND   BSR   COMPAR   GET RESULT
02343 34FC 8C 0000   CPX   #$0000   COMPARE WITH "FALSE"
02344 34FF 27 25   BEQ   EQUAL2   IF FALSE THEN LEAVE
02345 3501 FE 209C   LDX   OPRND1   GET OPRND1
02346 3504 27 20   BEQ   EQUAL2   IF FALSE THEN LEAVE ELSE
02347 3506 20 1B   BRA   EQUAL1   BOTH TRUE SO ANSWER = TRUE
02348      *
02349      *
02350      *           OR
02351      *           --
02352      *
02353      *   THIS ROUTINE PERFORMS THE BOOLEAN 'OR' ON TWO
02354      *   NUMBERS, NAMELY RESULT AND OPRND1 ACCORDING TO THE
02355      *   FOLLOWING TABLE :
02356      *
02357      *   RESULT      | OPRND1      | RESULT OR OPRND1
02358      *   -----+-----+-----
02359      *   0000      | 0000      | 0000
02360      *   0000      | OTHERWISE  | 0001
02361      *   OTHERWISE  | 0000      | 0001
02362      *   OTHERWISE  | OTHERWISE  | 0001
02363      *
02364      * I/P :
02365      *   RESULT = FIRST NUMBER
02366      *           = 0000 IF FALSE
02367      *           = OTHERWISE IF TRUE
02368      *   OPRND1 = SECOND NUMBER
02369      *           = 0000 IF FALSE
02370      *           = OTHERWISE IF TRUE
02371      *
02372      * O/P :
02373      *   RESULT = BOOLEAN ANSWER
02374      *           = 0000 IF FALSE
02375      *           = 0001 IF TRUE
02376      *
02377      * *****
02378 3508 8D 2C   OR   BSR   COMPAR   GET RESULT
02379 350A 8C 0000   CPX   #$0000   COMPARE WITH "FALSE"
02380 350D 26 14   BNE   EQUAL1   IF TRUE THEN ANSWER = TRUE
02381 350F FE 209C   LDX   OPRND1   GET OPRND1

```

```

02382 3512 26 0F      BNE      EQUAL1  IF TRUE THEN ANSWER = TRUE
02383 3514 20 10      BRA      EQUAL2  ELSE BOTH FALSE SO LEAVE
02384                *
02385                *
02386                *          NOT
02387                *          ---
02388                *
02389                *      THIS ROUTINE PERFORMS THE BOOLEAN COMPLEMENT ON
02390                *      A SINGLE NUMBER, NAMELY RESULT, ACCORDING TO THE
02391                *      FOLLOWING TABLE :
02392                *
02393                *      RESULT      | NOT RESULT
02394                *      -----+-----
02395                *      0000      | 0001
02396                *      OTHERWISE | 0000
02397                *
02398                * I/P :
02399                *      RESULT = NUMBER TO BE COMPLEMENTED
02400                *              = 0000 IF FALSE
02401                *              = OTHERWISE IF TRUE
02402                *
02403                * O/P :
02404                *      RESULT = COMPLEMENTED ANSWER
02405                *              = 0000 IF FALSE
02406                *              = 0001 IF TRUE
02407                *
02408                * *****
02409 3516 8D 1E      NOT      BSR      COMPAR  GET RESULT
02410 3518 8C 0000      CPX      #$0000  COMPARE WITH "FALSE"
02411 351B 26 09      BNE      EQUAL2  IF TRUE THEN MAKE FALSE
02412 351D 27 04      BEQ      EQUAL1  IF FALSE THEN MAKE TRUE
02413                *
02414                *
02415                *          EQUALS
02416                *          -----
02417                *
02418                *      THIS ROUTINE COMPARES TWO NUMBERS, NAMELY THE
02419                *      RESULT AND OPRND1 AND DETERMINES WHETHER THEY ARE
02420                *      EQUAL OR NOT.
02421                *      I.E. COMPARE:  RESULT = OPRND1
02422                *
02423                * I/P :
02424                *      RESULT = FIRST NUMBER
02425                *      OPRND1 = SECOND NUMBER
02426                *
02427                * O/P :
02428                *      RESULT = INDICATOR WHETHER NUMBERS ARE EQUAL OR
02429                *              NOT
02430                *              = 0000 IF FIRST NUMBER NOT = SECOND NUM
02431                *              = 0001 IF FIRST NUMBER = SECOND NUMBER
02432                *
02433                * *****

```

```

02434 351F 8D 15  EQUALS BSR  COMPAR  COMPARE TWO NUMBERS
02435 3521 26 03      BNE  EQUAL2  IF NUMBERS ARE SAME
02436 3523 7C 20A3  EQUAL1 INC  RESULT+1  SET RELATION TO BE TRUE
02437 3526 FE 20B3  EQUAL2 LDX  XTEMP0  RESTORE IX
02438 3529 39      RTS
02439      *
02440      *
02441      *          LESS THAN
02442      *          -----
02443      *
02444      *          THIS ROUTINE COMPARES TWO NUMBERS, NAMELY
02445      *          RESULT AND OPRND1 AND DETERMINES WHETHER RESULT
02446      *          IS LESS THAN OPRND1 OR NOT.
02447      *          I.E. COMPARE:  RESULT < OPRND1
02448      *
02449      *          I/P :
02450      *          RESULT = FIRST NUMBER
02451      *          OPRND1 = SECOND NUMBER
02452      *
02453      *          O/P :
02454      *          RESULT = INDICATOR WHETHER FIRST NUMBER IS LESS
02455      *          THAN SECOND NUMBER OR NOT
02456      *          = 0000 IF FIRST NUMBER NOT < SECOND NUM
02457      *          = 0001 IF FIRST NUMBER < SECOND NUMBER
02458      *
02459      *          *****
02460 352A 8D 0A  LT  BSR  COMPAR  COMPARE TWO NUMBERS
02461 352C 2C F9      BGE  EQUAL2  IF FIRST < SECOND THEN
02462 352E 20 F4      BRA  EQUAL1  SET RELATION TO BE TRUE
02463      *
02464      *
02465      *          GREATER THAN
02466      *          -----
02467      *
02468      *          THIS ROUTINE COMPARES TWO NUMBERS, NAMELY
02469      *          RESULT AND OPRND1 AND DETERMINES WHETHER RESULT
02470      *          IS GREATER THAN OPRND1 OR NOT.
02471      *          I.E. COMPARE:  RESULT > OPRND1
02472      *
02473      *          I/P :
02474      *          RESULT = FIRST NUMBER
02475      *          OPRND1 = SECOND NUMBER
02476      *
02477      *          O/P :
02478      *          RESULT = INDICATOR WHETHER FIRST NUMBER IS
02479      *          GREATER THAN SECOND NUMBER OR NOT
02480      *          = 0000 IF FIRST NUMBER NOT > SECOND NUM
02481      *          = 0001 IF FIRST NUMBER > SECOND NUMBER
02482      *
02483      *          *****
02484 3530 8D 04  GT  BSR  COMPAR  COMPARE TWO NUMBERS
02485 3532 2F F3      BLE  EQUAL2  IF FIRST > SECOND THEN

```

```

02486 3534 20 EE      BRA      EQUAL1  SET RELATION TO BE TRUE
02487                *
02488                *
02489                *      COMPARE
02490                *      -----
02491                *
02492                *      THIS ROUTINE COMPARES TWO NUMBERS IN
02493                *      PREPARATION FOR A RELATIONAL OPERATION.
02494                *
02495                *      I/P :
02496                *      RESULT = FIRST NUMBER TO BE COMPARED
02497                *      OPRND1 = SECOND NUMBER TO BE COMPARED
02498                *
02499                *      O/P :
02500                *      RESULT = INDICATOR INITIALISED TO FALSE
02501                *      RELATION EXPRESSION I.E.:
02502                *      = 0000
02503                *
02504                *      *****
02505 3536 FF 20B3 GQMPAR STX      XTEMP0  SAVE IX
02506 3539 FE 20A2      LDX      RESULT  READ RESULT INTO IX
02507 353C 7F 20A2      CLR      RESULT  SET RELATION TO BE
02508 353F 7F 20A3      CLR      RESULT+1 FALSE INITIALLY
02509 3542 BC 209C      CPX      OPRND1
02510 3545 26 01      BNE      COMPA1  EMULATE "CPX OPRND1"
02511 3547 39      RTS
02512 3548 FF 20AC COMPA1 STX      TEMP    TO MAKE COMPARISON
02513 354B B6 20AD      LDA A  TEMP+1
02514 354E B0 209D      SUB A  OPRND1+1 (SEE DUDCPX ROUTINE FOR
02515 3551 F6 20AC      LDA B  TEMP
02516 3554 F2 209C      SBC B  OPRND1  REASON AND DETAILS)
02517 3557 7E 2F85      JMP      DUDCP1
02518                *
02519                *
02520                *      PEEK
02521                *      ----
02522                *
02523                *      THIS ROUTINE DETERMINES THE MEMORY BYTE STORED
02524                *      AT THE ADDRESS SPECIFIED AS THE ONLY PARAMETER OF
02525                *      THE PEEK FUNCTION, AND RETURNS THIS BYTE IN
02526                *      DECIMAL NOTATION.
02527                *
02528                *      I/P :
02529                *      XTEMP1 = POINTER TO FIRST CHAR AFTER KEYWORD
02530                *
02531                *      O/P :
02532                *      RESULT = VALUE OF BYTE AT SPECIFIED ADDRESS
02533                *      XTEMP1 = POINTER TO CLOSING BRACKET OF FUNCTION
02534                *
02535                *      *****
02536 355A FE 20B5 PEEK  LDX      XTEMP1  RETRIEVE LINE POINTER
02537 355D 08      INX      MOVE LINE PTR PAST KEYWORD

```

```

02538 355E BD 377A      JSR    SKSPSC    MOVE PTR TO NEXT NONSPACE
02539 3561 86 28        LDA A  #'(      IF NEXT CHAR IS
02540 3563 A1 00        CMP A  0,X
02541 3565 27 03        BEQ    PEEK1     NOT AN OPEN BRACKET: "("
02542 3567 BD 319E      JSR    SYNERR    THEN SYNTAX ERROR OTHERWISE
02543 356A 08          PEEK1 INX         MOVE PTR PAST BRACKET
02544 356B BD 377A      JSR    SKSPSC    MOVE PTR TO NEXT NONSPACE
02545 356E FF 20B5      STX    XTEMP1    STORE LINE PTR
02546 3571 BD 320B      JSR    PRCEXP    PROCESS EXPRSN (RECURSION!)
02547 3574 FE 20A2      LDX    RESULT    GET ADDRESS OF BYTE
02548 3577 A6 00        LDA A  0,X      GET VALUE OF BYTE
02549 3579 7F 20A2      CLR    RESULT
02550 357C B7 20A3      STA A  RESULT+1  AND STORE IT IN "RESULT"
02551 357F 39          RTS
02552
02553
02554
02555
02556
02557
02558
02559
02560
02561
02562
02563
02564
02565
02566
02567
02568
02569
02570
02571
02572
02573
02574
02575
02576 3580 5F          FINDLN CLR B      ANTICIPATE LINE WILL BE FOUND
02577 3581 7D 2091      TST    LINENO    IF IN EDITOR/IMMEDIATE MODE
02578 3584 26 05        BNE    FINDL1    THEN GO AND INITIALISE
02579 3586 7D 2091      TST    LINENO    FIND POINTER AT START
02580 3589 27 12        BEQ    FINDL2    OF USER PROGRAM
02581 358B B6 2096      FINDL1 LDA A  MSPART IF IN RUN MODE THEN
02582 358E B1 2091      CMP A  LINENO    GO AND INITIALIZE FIND
02583 3591 22 0F        BHI    FINDL3    POINTER AT THE NEXT LINE ONLY
02584 3593 26 08        BNE    FINDL2    IF LINE LOOKING FOR IS
02585 3595 B6 2095      LDA A  LSPART    GREATER THAN CURRENT ONE
02586 3598 B1 2092      CMP A  LINENO+1  OTHERWISE GO AND INITIALIZE
02587 3598 22 05        BHI    FINDL3    FIND POINTER AT PROG START
02588 359D CE 1001      FINDL2 LDX    #PROGST SET FIND PTR AT PROG START
02589 35A0 20 03        BRA     FINDL4

```

```

02590 35A2 FE 20A4 FINDL3 LDX      RUNPTR    SET FIND PTR AT NEXT LINE
02591 35A5 BC 20A0 FINDL4 CPX      PROGEN     IF END OF USERS PROGRAM
02592 35A8 27 12          BEQ      NOTFND     THEN LINE NOT FOUND
02593 35AA B6 2096          LDA A   MSPART     COMPARE LINE NO. LOOKING FOR
02594 35AD A1 00          CMP A   0,X        WITH LINE NO. AT FIND PTR
02595 35AF 22 10          BHI      SEARCH     IF THEY ARE THE SAME
02596 35B1 26 09          BNE      NOTFND     THEN LINE FOUND, IF
02597 35B3 B6 2095          LDA A   LSPART     LINE NO. AT FIND PTR IS LESS
02598 35B6 A1 01          CMP A   1,X        THEN CONTINUE SEARCH, IF LINE
02599 35B8 22 07          BHI      SEARCH     NO. AT FIND POINTER IS MORE
02600 35BA 27 01          BEQ      FOUND      THEN LINE NOT FOUND
02601 35BC 53          NOTFND COM B          LINE NOT FOUND: SET FLAG
02602 35BD FF 20BD FOUND  STX      FNDPTR     LINE FOUND: STORE FIND PTR
02603 35C0 39          RTS                    AND EXIT
02604 35C1 E6 02          SEARCH LDA B   2,X    CONTINUE SEARCHING:
02605 35C3 BD 375D          JSR      IXADD     MOVE FIND POINTER TO
02606 35C6 5F          CLR B                    BEGINNING OF NEXT LINE
02607 35C7 20 DD          BRA      FINDL4     COMPARE LINE NO.S AGAIN
02608 *
02609 *
02610 *          GET VARIABLE
02611 *          -----
02612 *
02613 *          THIS ROUTINE GETS THE VARIABLE SPECIFIED IN
02614 * VARNAM AND PLACES IT IN VAR READY FOR THE GET
02615 * VARIABLE VALUE ROUTINE.
02616 *
02617 * I/P :
02618 * VARNAM = VARIABLE NAME
02619 *
02620 * O/P :
02621 * VAR = SAME VARIABLE NAME
02622 *
02623 * *****
02624 35C9 F6 20B2 GETVAR LDA B   VARNAM    GET VARIABLE NAME AND
02625 35CC F7 20B1 STA B   VAR      STORE IT IN VAR
02626 *
02627 *
02628 *          GET VARIABLE VALUE
02629 *          -----
02630 *
02631 *          THIS ROUTINE RETURNS THE CURRENT VALUE FOR THE
02632 * VARIABLE SPECIFIED IN VAR.
02633 *
02634 * I/P :
02635 * VAR = VARIABLE WHOSE VALUE IS REQUIRED
02636 *
02637 * O/P :
02638 * IX = VALUE OF VARIABLE: VAR
02639 *
02640 * *****
02641 35CF 8D 03 GETVAL BSR      GETVAL    GET VARIABLE ADDRESS

```

```

02642 35D1 EE 00          LDX      0,X      GET VARIABLE VALUE
02643 35D3 39             RTS
02644 35D4 F6 20B1 GETVAL LDA B  VAR      CALCULATE OFFSET FOR
02645 35D7 C0 41          SUB B   #'A      VARIABLE NAME FROM
02646 35D9 58             ASL B           BASE ADDRESS OF
02647 35DA CE 2000        LDX      #VARBEG  VARIABLE AREA
02648 35DD BD 375D        JSR      IXADD   ADD OFFSET TO BASE ADDR
02649 35E0 39             RTS
02650
02651
02652
02653
02654
02655
02656
02657
02658
02659
02660
02661
02662
02663
02664
02665
02666
02667
02668
02669
02670 35E1 7F 2096 NUMBCD CLR      MSPART  INITIALIZE LINE NUM = 0000
02671 35E4 7F 2095          CLR      LSPART
02672 35E7 86 04          LDA A   #4      SET UP COUNT OF CHARS TO READ
02673 35E9 B7 207F        STA A   COUNT
02674 35EC BD 3746 NUMBC1 JSR      CLASS   CLASSIFY CURRENT CHAR
02675 35EF C1 01          CMP B   #s01   IF CHAR IS NOT A DIGIT
02676 35F1 26 1B          BNE      ENDNUM  THEN END READING NO., SO EXIT
02677 35F3 86 04          LDA A   #4      OTHERWISE
02678 35F5 78 2095 NUMBC2 ASL      LSPART  MULTIPLY LINE NUMBER
02679 35F8 79 2096        ROL      MSPART
02680 35FB 4A             DEC A           BY 10
02681 35FC 26 F8          BNE      NUMBC2
02682 35FE A6 00          LDA A   0,X     GET CURRENT CHARACTER
02683 3600 84 0F          AND A   #LSMASK REMOVE UPPER HALF OF BYTE
02684 3602 BA 2095        ORA A   LSPART  AND ADD RESULT TO THE
02685 3605 B7 2095        STA A   LSPART  LINE NUMBER AT THIS STAGE
02686 3608 08             INX
02687 3609 7A 207F        DEC      COUNT  MOVE POINTER TO NEXT CHAR
02688 360C 26 DF          BNE      NUMBC1  DEC COUNT OF CHARS TO BE READ
02689 360E 39             ENDNUM RTS      IF MORE TO READ GO READ NEXT
02690
02691
02692
02693

```

CONVERT BCD TO NUMBER

```

02694      *
02695      *   THIS ROUTINE CONVERTS A 2 BYTE PACKED BCD LINE
02696      *   NUMBER BACK TO A 4 BYTE NUMERIC FORMAT.
02697      *
02698      * I/P :
02699      *   IX = POINTER TO BCD LINE NUMBER
02700      *
02701      * O/P :
02702      *   ERRLIN = 4 BYTE ASCII CONVERSION OF LINE NO.
02703      *
02704      *
02705      *****
02705 360F FF 20B3 BCDNUM STX   XTEMP0  STORE ADDR OF BCD LINE NO.
02706 3612 CE 2087      LDX   #ERRLIN  SET UP ADDR OF CONVERTED
02707 3615 FF 20B5      STX   XTEMP1  NUMBER
02708 3618 7F 208C      CLR   FLAG    SET UP FLAG
02709 361B FE 20B3 BCDNU1 LDX   XTEMP0  RETRIEVE ADDR OF BCD LINE NO.
02710 361E A6 00      LDA   A    0,X   TAKE BYTE FROM
02711 3620 84 F0      AND   A    #MSMASK
02712 3622 44      LSR   A          BCD LINE NUMBER AND
02713 3623 44      LSR   A
02714 3624 44      LSR   A          CALCULATE MS DIGIT
02715 3625 44      LSR   A
02716 3626 8A 30      ORA   A    #$30   FROM IT
02717 3628 E6 00      LDA   B    0,X   TAKE BYTE FROM BCD LINE
02718 362A C4 0F      AND   B    #LSMASK  NUMBER AND CALCULATE
02719 362C CA 30      ORA   B    #$30   LS DIGIT FROM IT
02720 362E 08      INX          BUMP POINTER TO NEXT BYTE
02721 362F FF 20B3      STX   XTEMP0  AND STORE POINTER
02722 3632 FE 20B5      LDX   XTEMP1  RETRIEVE DESTINATION ADDR
02723 3635 A7 00      STA   A    0,X   WRITE MS DIGIT TO DESTINATION
02724 3637 E7 01      STA   B    1,X   WRITE LS DIGIT TO DESTINATION
02725 3639 08      INX          BUMP POINTER TO NEXT
02726 363A 08      INX          LOCATIONS IN DESTINATION
02727 363B FF 20B5      STX   XTEMP1  STORE DESTINATION POINTER
02728 363E 73 208C      COM   FLAG    INVERT FLAG AND
02729 3641 26 D9      BNE   BCDNU1  REPEAT ONCE MORE,
02730 3643 39      RTS          THEN EXIT
02731      *
02732      *
02733      *   CONVERT HEX NUMBER TO ASCII
02734      *   -----
02735      *
02736      *   THIS ROUTINE CONVERTS A 2'S COMPLEMENT HEX
02737      *   NUMBER TO A 5 DIGIT (5 BYTE) ASCII EQUIVALENT,
02738      *   WITH THE SIGN INDICATED SEPARATELY IN A FLAG.
02739      *
02740      * I/P :
02741      *   RESULT = HEX NUMBER TO BE CONVERTED
02742      *
02743      * O/P :
02744      *   NUMBER = 5 DIGIT ASCII CONVERSION
02745      *   SIGN = SIGN OF ASCII CONVERSION

```



```

02746      *      = 00    IF POSITIVE
02747      *      = FF    IF NEGATIVE
02748      *
02749      *****
02750 3644 7F 20A6 HEXASC CLR    SIGN    SET SIGN = POSITIVE NO.
02751 3647 FE 20A2      LDX    RESULT  IF NUMBER TO BE
02752 364A 8C 8000      CPX    #$8000
02753 364D 26 15      BNE    HEXAS1    CONVERTED IS -32768
02754 364F 73 20A6      COM    SIGN
02755 3652 CE 3332      LDX    #$3332    THEN SPECIAL CASE :
02756 3655 FF 2097      STX    NUMBER
02757 3658 CE 3736      LDX    #$3736    MAKE SIGN NEGATIVE AND
02758 365B FF 2099      STX    NUMBER+2
02759 365E 86 38      LDA A    #$38    SET UP DIGITS IN NUMBER
02760 3660 B7 209B      STA A    NUMBER+4
02761 3663 39      RTS
02762 3664 B6 20A2 HEXAS1 LDA A    RESULT    LOOK TO SEE IF
02763 3667 84 80      AND A    #NEGMSK    NUMBER IS NEGATIVE
02764 3669 27 09      BEQ    HEXAS2    IF SO THEN
02765 366B CE 20A2      LDX    #RESULT
02766 366E BD 3415      JSR    NEGAT1    NEGATE NUMBER
02767 3671 73 20A6      COM    SIGN
02768 3674 CE 2097 HEXAS2 LDX    #NUMBER    SET PTR AT MS DIGIT
02769 3677 FF 20B9      STX    XTEMP3    OF ASCII NUMBER
02770 367A CE 37EE      LDX    #DECTBL    SET TBL PTR START DECTBL
02771 367D A6 00      HEXAS3 LDA A    0,X    READ DECIMAL NUMBER
02772 367F B7 209C      STA A    OPRND1    AT DECIMAL TABLE
02773 3682 A6 01      LDA A    1,X    POINTER AND SET NUMBER
02774 3684 B7 209D      STA A    OPRND1+1    UP IN OPRND1
02775 3687 F6 20A6      LDA B    SIGN    SAVE SIGN
02776 368A BD 3494      JSR    DIVIDE    DIVIDE RESULT BY OPRND1
02777 368D F7 20A6      STA B    SIGN    RESTORE SIGN
02778 3690 B6 20A3      LDA A    RESULT+1    GET SINGLE DIGIT ANSWER
02779 3693 8A 30      QRA A    #$30    CONVERT TO ASCII CHAR
02780 3695 FF 20B3      STX    XTEMP0    SAVE DECIMAL TABLE PTR
02781 3698 FE 20B9      LDX    XTEMP3    RETRIEVE ASCII NO. PTR
02782 369B A7 00      STA A    0,X    STORE ASCII DIGIT
02783 369D 08      INX    MOVE PTR TO NEXT ASCII DIGIT
02784 369E FF 20B9      STX    XTEMP3    STORE ASCII NO. PTR
02785 36A1 FE 209C      LDX    OPRND1    GET REMAINDER FROM DIVISION
02786 36A4 FF 20A2      STX    RESULT    AND SET UP IN RESULT
02787 36A7 FE 20B3      LDX    XTEMP0    RETRIEVE DECIMAL TABLE PTR
02788 36AA 08      INX    MOVE DECIMAL TABLE PTR
02789 36AB 08      INX    TO NEXT DECIMAL NUMBER
02790 36AC 8C 37F8      CFX    #DECTBL+10    IF NOT FINISHED CONVERSIO
02791 36AF 26 CD      BNE    HEXAS3    THEN REPEAT
02792 36B1 39      RTS
02793      *
02794      *
02795      *      CONVERT ASCII NUMBER TO HEX
02796      *      -----
02797      *

```

```

02798      *      THIS ROUTINE READS AND CONVERTS AN ASCII SIGNE
02799      *      INTEGER TO ITS 2 BYTE, 2'S COMPLEMENT HEX
02800      *      EQUIVALENT. THE ASCII NUMBER MAY INCLUDE AN
02801      *      OPTIONAL SIGN (+ OR -).
02802      *
02803      *      I/P :
02804      *      IX = PTR TO START OF NUMBER TO BE CONVERTED
02805      *      FLAG = INDICATOR WHERE THIS ROUTINE CALLED FROM
02806      *      = FF   IF CALLED FROM INPUT
02807      *      = 00   OTHERWISE
02808      *
02809      *      O/P :
02810      *      IX = PTR TO FIRST CHAR AFTER NUMBER CONVERTED
02811      *      RESULT = 2 BYTE HEX CONVERSION
02812      *
02813      *****
02814 36B2 7F 20A6 ASCHEX CLR      SIGN      SET SIGN = POSITIVE NO.
02815 36B5 A6 00      LDA A      0,X        IF FIRST CHAR IS
02816 36B7 81 2D      CMP A      #'-'      A "-" THEN
02817 36B9 26 05      BNE      ASCHX1
02818 36BB 73 20A6      COM      SIGN      SET SIGN = NEGATIVE NO.
02819 36BE 20 04      BRA      ASCHX2
02820 36C0 81 2B      ASCHX1 CMP A      #'+'      IF FIRST CHAR IS
02821 36C2 26 04      BNE      ASCHX3      A SIGN CHAR THEN
02822 36C4 08      ASCHX2 INX          MOVE POINTER PAST SIGN
02823 36C5 BD 377A      JSR      SKPSPEC SKIP TO NEXT NONSPACE
02824 36C8 BD 3746 ASCHX3 JSR      CLASS   IF NEXT CHAR
02825 36CB C1 01      CMP B      #$01      NOT NUMERIC THEN :
02826 36CD 27 0D      BEQ      ASCHX5
02827 36CF 7D 208C      TST      FLAG
02828 36D2 27 05      BEQ      ASCHX4      BAD INPUT ERROR IF
02829 36D4 32      PUL A          ASCHX4 CALLED FROM INPUT
02830 36D5 32      PUL A          SO UNSTACK LAST RETURN ADDR
02831 36D6 7E 31D1      JMP      BADINP    AND DISPLAY MESSAGE, ELSE
02832 36D9 7E 319E ASCHX4 JMP      SYNERR  DISPLAY SYNTAX ERROR MESSAGE
02833 36DC 7F 20A2 ASCHX5 CLR      RESULT SET HEX NO. TO
02834 36DF 7F 20A3      CLR      RESULT+1 ZERO INITIALLY
02835 36E2 7F 209C ASCHX6 CLR      OPRND1
02836 36E5 A6 00      LDA A      0,X        GET A DIGIT
02837 36E7 84 0F      AND A      #LSMASK  REMOVE MS PART OF BYTE
02838 36E9 B7 209D      STA A      OPRND1+1 SET UP OPERAND 1 FROM DIGIT
02839 36EC 8D 2F      BSR      ADDOP     ADD OPERAND 1 TO RESULT
02840 36EE 08      INX          MOVE POINTER TO NEXT CHAR
02841 36EF BD 3746      JSR      CLASS   IF NEXT CHAR IS
02842 36F2 C1 01      CMP B      #$01      ANOTHER NUMERIC THEN
02843 36F4 26 15      BNE      ASCHX8
02844 36F6 B6 20A2      LDA A      RESULT
02845 36F9 B7 209C      STA A      OPRND1  MULTIPLY RESULT
02846 36FC B6 20A3      LDA A      RESULT+1
02847 36FF B7 209D      STA A      OPRND1+1 BY 10 USING
02848 3702 C6 09      LDA B      #9
02849 3704 BD 17      ASCHX7 BSR      ADDOP  REPEATED ADDITION

```

```

02850 3706 5A          DEC B
02851 3707 26 FC      BNE  ASCHX7
02852 3709 20 D8      BRA  ASCHX6   REPEAT FOR NEXT DIGIT
02853 370B 7D 20A6 ASCHX8 TST  SIGN   IF SIGN IS NEGATIVE
02854 370E 27 0C      BEQ  ASCHX9
02855 3710 FF 20B3     STX  XTEMP0   THEN NEGATE RESULT
02856 3713 CE 20A2     LDX  #RESULT
02857 3716 BD 3415     JSR  NEGAT1   IN 2'S COMPLEMENT
02858 3719 FE 20B3     LDX  XTEMP0
02859 371C 39          ASCHX9 RTS      RETURN
02860 371D B6 20A3 ADDOP LDA A  RESULT+1  ADD LOWER ORDER
02861 3720 BB 209D     ADD A  OPRND1+1  BYTES TOGETHER
02862 3723 B7 20A3     STA A  RESULT+1  STORE BACK IN RESULT
02863 3726 B6 20A2     LDA A  RESULT    ADD HIGHER ORDER BYTES
02864 3729 B9 209C     ADC A  OPRND1    INCLUDING ANY CARRY FROM
02865 372C 28 14      BVC  ADDOP2     PREVIOUS ADDITION
02866 372E 81 80      CMP A  #$80     IF ARITHMETIC OVERFLOW
02867 3730 26 0A      BNE  ADDOP1
02868 3732 7D 20A3     TST  RESULT+1   THEN DISPLAY ERROR MESSAGE
02869 3735 26 05      BNE  ADDOP1
02870 3737 7D 20A6     TST  SIGN        BUT ONLY IF RESULT IS NOT
02871 373A 26 06      BNE  ADDOP2
02872 373C CE 4F56 ADDOP1 LDX  #ARITOV  $8000 WITH NEGATIVE SIGN
02873 373F 7E 31A1     JMP  ERROR      (I.E. : -32768)
02874 3742 B7 20A2 ADDOP2 STA A  RESULT  STORE BACK IN RESULT
02875 3745 39          RTS
02876          *
02877          *
02878          *      CLASSIFY A CHARACTER
02879          *      -----
02880          *
02881          *      THIS ROUTINE DETERMINES THE CLASS OF A
02882          *      CHARACTER POINTED TO BY THE INDEX REGISTER. THE
02883          *      CLASS IS RETURNED IN ACCUMULATOR B.
02884          *
02885          *      I/P :
02886          *      IX * POINTER TO CHARACTER TO BE CLASSIFIED
02887          *
02888          *      O/P :
02889          *      ACCB = CLASS OF CHARACTER :
02890          *      = 01 FOR CHARACTERS 0...9
02891          *      = 02 FOR CHARACTERS A...Z
02892          *      = 03 FOR OTHER CHARACTERS
02893          *
02894          *      *****
02895 3746 A6 00 CLASS LDA A  0,X      GET CHARACTER
02896 3748 C6 01          LDA B  #$01    INITIALIZE CHAR TYPE = DIGIT
02897 374A 80 30          SUB A  #'0     IS CHAR BETWEEN "0"
02898 374C 2D 04          BLT  CLASS1
02899 374E 80 0A          SUB A  #10     AND "9"
02900 3750 2D 0A          BLT  CLASS3    IF SO THEN EXIT OTHERWISE
02901 3752 5C          CLASSI INC B      SET CHAR TYPE = LETTER

```

```

02902 3753 80 07      SUB A  #7      IS CHAR TYPE
02903 3755 2D 04      BLT    CLASS2  A LETTER ?
02904 3757 80 1A      SUB A  #26
02905 3759 2D 01      BLT    CLASS3  IF S0 THEN EXIT OTHERWISE
02906 375B 5C          CLASS2 INC B  SET CHAR TYPE = OTHER
02907 375C 39          CLASS3 RTS
02908                  *
02909                  *
02910                  *      ADD TO IX
02911                  *      -----
02912                  *
02913                  *      THIS ROUTINE ADDS A SPECIFIED QUANTITY TO THE
02914                  *      INDEX REGISTER.
02915                  *
02916                  *      I/P :
02917                  *      IX = ORIGINAL VALUE OF IX
02918                  *      ACCB = AMOUNT TO BE ADDED TO IX
02919                  *
02920                  *      O/P :
02921                  *      IX = RESULTING VALUE OF IX
02922                  *
02923                  *      *****
02924 375D 5D          IXADD  TST B      IF ACCB NOT = 00
02925 375E 27 04      BEQ    IXADD1
02926 3760 08          INX
02927 3761 5A          DEC B      ADD 1 TO IX
02928 3762 20 FA      BRA    IXADD    REPEAT UNTIL FINISHED
02929 3764 39          IXADD1 RTS
02930                  *
02931                  *
02932                  *      SUBTRACT FROM IX
02933                  *      -----
02934                  *
02935                  *      THIS ROUTINE SUBTRACTS A SPECIFIED QUANTITY
02936                  *      FROM THE INDEX REGISTER.
02937                  *
02938                  *      I/P :
02939                  *      IX = ORIGINAL VALUE OF IX
02940                  *      ACCB = AMOUNT TO BE SUBTRACTED FROM IX
02941                  *
02942                  *      O/P :
02943                  *      IX = RESULTING VALUE OF IX
02944                  *
02945                  *      *****
02946 3765 5D          IXSUB  TST B      IF ACCB NOT = 00
02947 3766 27 04      BEQ    IXSUB1
02948 3768 09          DEX
02949 3769 5A          DEC B      SUBTRACT 1 FROM IX
02950 376A 20 FA      BRA    IXSUB    REPEAT UNTIL FINISHED
02951 376C 39          IXSUB1 RTS
02952                  *
02953                  *

```

```

02954      *          FIND SPACE
02955      *          -----
02956      *
02957      *          THIS ROUTINE SEARCHES FOR THE NEXT SPACE
02958      * CHARACTER, STOPPING ITS SEARCH ON REACHING THE END
02959      * OF A LINE I.E. ON ENCOUNTERING A <CR>. ON ENDING
02960      * THE SEARCH FOR A SPACE THE SKIP SPACE ROUTINE IS
02961      * THEN EXECUTED TO MOVE THE POINTER TO THE NEXT
02962      * NONSPACE CHARACTER.
02963      *
02964      * I/P :
02965      * IX = ORIGINAL VALUE FOR POINTER
02966      *
02967      * O/P :
02968      * IX = POINTER TO FIRST SPACE OR <CR> FOUND
02969      *
02970      *
02971 376D A6 00  FNDSPC LDA A 0,X      READ CHARACTER AT POINTER
02972 376F 81 20      CMP A #$20      IF CHARACTER IS A SPACE
02973 3771 27 07      BEQ  SKPSPC      THEN EXIT OTHERWISE
02974 3773 81 0D      CMP A #CR      IF END OF LINE
02975 3775 27 03      BEQ  SKPSPC      THEN EXIT OTHERWISE
02976 3777 08        INX             INCREMENT POINTER TO NEXT
02977 3778 20 F4      BRA  FNDSPC      CHARACTER AND REPEAT
02978      *
02979      *
02980      *          SKIP SPACES
02981      *          -----
02982      *
02983      *          THIS ROUTINE SKIPS TO THE FIRST NONSPACE
02984      * CHARACTER ENCOUNTERED IF NOT ALREADY AT A NONSPACE
02985      * CHARACTER.
02986      *
02987      * I/P :
02988      * IX = ORIGINAL VALUE FOR POINTER
02989      *
02990      * O/P :
02991      * IX = POINTER TO FIRST NONSPACE FOUND
02992      * ACCA = FIRST NONSPACE CHARACTER FOUND
02993      *
02994      *
02995 377A A6 00  SKPSPC LDA A 0,X      IS CHAR AT IX A SPACE ?
02996 377C 81 20      CMP A #$20
02997 377E 26 03      BNE  SKPSP1
02998 3780 08        INX             IF SO SKIP TO NEXT CHAR
02999 3781 20 F8      BRA  SKPSPC
03000 3783 39      SKPSP1 RTS
03001      *
03002      *
03003      *          CHECK STACK
03004      *          -----
03005      *

```

```

03006      *      THIS ROUTINE CHECKS TO SEE IF ADDING ANOTHER
03007      *      ELEMENT (3 BYTES) TO THE USER'S STACK WILL CAUSE
03008      *      STACK OVERFLOW - IF SO A STACK OVERFLOW MESSAGE IS
03009      *      DISPLAYED.
03010      *
03011      *****
03012 3784 FF 20B3 CHKSTK STX      XTEMP0      SAVE IX
03013 3787 FE 20AF          LDX      USRSP      CHECK THAT STACK POINTER
03014 378A 8C 2700          CPX      #STKLIM    IS GREATER THAN THE STACK
03015 378D 26 06          BNE      STAKOK      LIMIT. IF NOT THEN
03016 378F CE 534F          LDX      #STAKOV    SET UP ERROR MESSAGE
03017 3792 7E 31A1          JMP      ERROR      AND DISPLAY ERROR MESSAGE
03018 3795 FE 20B3 STAKOK LDX      XTEMP0      OTHERWISE RETRIEVE IX
03019 3798 39              RTS              AND EXIT
03020      *
03021      *
03022      *      CHECK FOR <CR> #1
03023      *      -----
03024      *
03025      *      THIS ROUTINE CHECKS TO SEE IF THE NEXT
03026      *      NONSPACE CHARACTER IS A <CR> FOR OPERATING SYSTEM
03027      *      COMMANDS - IF NOT CONTROL IS TRANSFERRED TO THE
03028      *      "WHAT?" ROUTINE.
03029      *
03030      *      I/P :
03031      *      IX = POINTER TO LINE TO BE CHECKED
03032      *
03033      *****
03034 3799 BD 377A CHKCR1 JSR      SKPSPC      SKIP TO FIRST NONSPACE
03035 379C 81 0D          CMP A      #CR      IF THIS CHARACTER IS
03036 379E 27 0D          BEQ      CHKCR3      NOT A <CR> THEN
03037 37A0 7E 31C5          JMP      WHAT      DISPLAY WHAT MESSAGE
03038      *
03039      *
03040      *      CHECK FOR <CR> #2
03041      *      -----
03042      *
03043      *      THIS ROUTINE CHECKS TO SEE IF THE NEXT
03044      *      NONSPACE CHARACTER IS A <CR> FOR BASIC KEYWORDS
03045      *      - IF NOT CONTROL IS TRANSFERRED TO THE ERROR
03046      *      ROUTINE.
03047      *
03048      *      I/P :
03049      *      IX = POINTER TO LINE TO BE CHECKED
03050      *
03051      *****
03052 37A3 BD 377A CHKCR2 JSR      SKPSPC      SKIP TO FIRST NONSPACE
03053 37A6 81 0D          CMP A      #CR      IF THIS CHARACTER IS
03054 37A8 27 03          BEQ      CHKCR3      NOT A <CR> THEN
03055 37AA 7E 319E          JMP      SYNERR      DISPLAY SYNTAX ERROR MESSAGE
03056 37AD 39          CHKCR3 RTS              OTHERWISE EXIT
03057      *

```

```

03058      *
03059      *          <BREAK> INTERRUPT SERVICE ROUTINE
03060      *          -----
03061      *
03062      *          THIS ROUTINE SETS A BREAK FLAG TO INDICATE THAT
03063      *          THE <BREAK> KEY WAS PRESSED BY THE USER.
03064      *
03065      *          *****
03066 37AE 0F 207C BRKISR SEI      DISABLE ANY MORE INTERRUPTS
03067 37AF 73      COM          BRKFLG      SET <BREAK> FLAG
03068 37B2 0E      CLI          ENABLE INTERRUPTS AGAIN
03069 37B3 3B      RTI          RETURN FROM ISR
03070      *
03071      *
03072      *          MESSAGES
03073      *          -----
03074      *
03075      *          *****
03076 37B4 0D      STRTUP FCB    CR,LF,LF      STARTUP MESSAGE
          37B5 0A
          37B6 0A
03077 37B7 3C      FCC          '<BASIC>'
          37B8 42
          37B9 41
          37BA 53
          37BB 49
          37BC 43
          37BD 3E
03078 37BE 0D      FCC          CR,LF,LF,EOT
          37BF 0A
          37C0 0A
          37C1 04
03079 37C2 0D      MSG1 FCB    CR,LF      OPERATING SYSTEM PROMPT
          37C3 0A
03080 37C4 21      FCC          '!'
          37C5 20
03081 37C6 04      FCC          EOT
03082      773A      MSG2 EQU    $773A      "7" INPUT PROMPT
03083 37C7 20      MSG3 FCC      ' ERROR IN ' ERROR MESSAGE
          37C8 45
          37C9 52
          37CA 52
          37CB 4F
          37CC 52
          37CD 20
          37CE 49
          37CF 4E
          37D0 20
03084 37D1 04      FCC          EOT
03085 37D2 0D      MSG4 FCB    CR,LF      <BREAK>/STOP MESSAGE
          37D3 0A
03086 37D4 53      FCC          'STOPPED AT '

```

```

37D5 54
37D6 4F
37D7 50
37D8 50
37D9 45
37DA 44
37DB 20
37DC 41
37DD 54
37DE 20
03087 37DF 04          FCB      EOT
03088 37E0 0D          MSG5    FCB      CR,LF      BAD INPUT MESSAGE
        37E1 0A
03089 37E2 42          FCC      'BAD INPUT'
        37E3 41
        37E4 44
        37E5 20
        37E6 49
        37E7 4E
        37E8 50
        37E9 55
        37EA 54
03090 37EB 0D          FCB      CR,LF,EOT
        37EC 0A
        37ED 04
03091      76CA          MSG6    EQU      $76CA      "WHAT?" ERROR MESSAGE
03092      *
03093      *
03094      *          TABLE OF DECIMAL NUMBERS
03095      *          -----
03096      *
03097      *          THE NUMBERS IN THIS TABLE EACH FILL 2 BYTES AND
03098      *          ARE IN 2'S COMPLEMENT FORM. THEY CONSIST OF THE
03099      *          DECIMAL NUMBERS :
03100      *          10000
03101      *          1000
03102      *          100
03103      *          10
03104      *          1
03105      *          I.E. EACH NUMBER IS A POWER OF 10. THIS TABLE IS
03106      *          USED IN THE ROUTINE "CONVERT HEX NUMBER TO ASCII".
03107      *
03108      *          *****
03109 37EE 27          DECTBL    FCB      $27,$10      DECIMAL 10000
        37EF 10
03110 37F0 03          FCB      $03,$E8      DECIMAL 1000
        37F1 E8
03111 37F2 00          FCB      $00,$64      DECIMAL 100
        37F3 64
03112 37F4 00          FCB      $00,$0A      DECIMAL 10
        37F5 0A
03113 37F6 00          FCB      $00,$01      DECIMAL 1

```


37F7 01

03114

03115

END

SYMBOL TABLE

RACKSP	0008	CR	000D	ENDADD	0010	EOT	0004	FILMEM	722F
IN	70DA	LF	000A	LSMASK	000F	MEMLIM	2000	MOVMEM	7252
MSMASK	00F0	NEGMSK	0080	NMIV	0000	NXTSPC	73CB	OUT	70E7
OUTS	70C5	PCRLF	7402	PROGST	1001	PSTRNG	703F	RESET	7008
STARAD	000E	STKLIM	2700	TABSIZ	0008	USRSTK	27FF	VARBEG	2000
VAREND	2033	XTEMP	0006	SYNTAX	534E	NOLINE	4E4C	DIVZER	2F30
ARITOV	4F56	FORNEX	464E	NEXFOR	4E46	GOSRET	4752	RETGOS	5247
OUTDAT	4F44	OUTMEM	4F4D	EXPERR	4545	STAKOV	534F	NOCONT	4343
BUFFER	2034	BRKFLG	207C	BUFFTR	207D	COUNT	207F	DATPTR	2080
ENDVAL	2082	ERRCOD	2084	ERRLIN	2087	FLAG	208C	FNDPTR	208D
LASTCH	208F	LENGTH	2090	LINENO	2091	LRGLIN	2093	LSPART	2095
MSPART	2096	NUMBER	2097	OPRND1	209C	PRGCNT	209E	PROGEN	20A0
RESULT	20A2	RUNPTR	20A4	SIGN	20A6	SP	20A7	TABPOS	20A9
TBLPTR	20AA	TEMP	20AC	UNOPTR	20AE	USRSP	20AF	VAR	20B1
VARNAM	20B2	XTEMP0	20B3	XTEMP1	20B5	XTEMP2	20B7	XTEMP3	20B9
XTEMP4	20BB	XTEMP5	20BD	START	2800	RESTRT	2803	INTLZE	280F
	281F	NEW1	2825	INTPRT	284E	EDMODE	2875	FILBUF	2879
FILBF1	287D	FILBF2	288C	FILBF3	2894	FILBF4	28A5	OPSYS	28A7
OPSYS1	288C	IMMEDI	28BF	IMMEDI	28D2	EDITOR	28D5	LINEOK	28E5
EDITR1	2903	EDITR2	290B	EDITR3	2920	EDITR4	2923	EDITR5	2926
INSERT	2929	APPEND	294E	DELETE	2962	DELET1	296B	DELET2	297E
DELET3	298D	DELET4	2993	UPDEND	29B1	UPDEN1	29B5	UPDEN2	29C4
UPDEN3	29D0	ADDLN	29D7	ADDLN1	29E9	FINDKE	2A00	FINDK1	2A03
FINDK2	2A05	FINDK3	2A12	FINDK4	2A22	FINDK5	2A30	FINDK6	2A33
COMTBL	2A3A	RUNTBL	2A4F	IMMTBL	2A6D	IFTBL	2A9B	FMTBL	2ABA
LIST	2AC0	LIST1	2AD8	LIST2	2AEE	LIST3	2AF4	OUTLIS	2AFA
OUTLS1	2B0A	OUTLS2	2B24	ENDLIS	2B3C	MONITR	2B3E	RUN	2B47
GORUN	2B7B	RUN1	2B7C	RUN2	2B86	RUN3	2B8F	EXECUT	2B90
EXECU1	2BAD	EXECU2	2BB0	EXECU3	2BB3	EXECU4	2BC2	EXECU5	2BCC
FND	2BCF	END1	2BD5	END2	2BDB	FOR	2BE1	FOR1	2BEE
FOR2	2BF1	FOR3	2C09	FOR4	2C1E	FOR5	2C23	FOR6	2C2E
FOR7	2C34	FOR9	2C6B	FOR10	2C6C	FOR11	2C71	TO	2C74
TO1	2C7F	TO2	2C82	STEP	2C94	STEP1	2CA7	STEP2	2CB1
MTFR3	2CB4	ENDFOR	2CDB	ENDFR1	2CE7	ENDFR2	2D0D	ENDFR3	2D1A
ENDFR4	2D20	ENDFR5	2D28	ENDFR6	2D30	GOSUB	2D39	GOTO	2D51
WTO1	2D61	GOTO2	2D67	GOTO3	2D73	GOTO4	2D8F	IF	2D90
IFI	2DAD	IF2	2DB0	THEN	2DC5	THEN1	2DDC	THEN2	2DE6
TRIPN1	2DE9	INPUT	2DF2	INPUT1	2DFB	INPUT2	2E02	INPUT3	2E05
INPUT4	2E0D	INPUT5	2E2B	INPUT6	2E49	INPUT7	2E4D	INPUT8	2E7A
EXPNSN	2E7E	EXPRS1	2E9A	EXPRS2	2EA2	EXPRS3	2EA5	EXPRS4	2EBE
EXPNS5	2EC1	PXTVAL	2EC7	NEXT	2ED5	NEXT1	2EE2	NEXT2	2EE5
NEXT1	2EF4	NEXT4	2F09	NEXT5	2F0E	NEXT6	2F14	NEXT7	2F1F
NEXT8	2F2F	NEXT9	2F3B	NEXT10	2F3E	NEXT11	2F5C	NEXT12	2F66
NEXT13	2F6A	DUDCPX	2F71	DUDCP1	2F85	DUDCP2	2F89	POKE	2F8A

POKE1	2FA5	PRINT	2FBB	PRINT1	2FC4	PRINT2	2FCB	NUMB	2FDA
NUMB1	2FEC	NUMB2	2FEF	NUMB3	2FF9	NUMB4	3001	NUMB5	3002
PRINT3	301B	PRINT4	302F	STRING	3032	STRNG1	3040	TAB	3047
NOSPC	3057	PRINT5	3061	PRINT6	3077	UPDTAB	3078	UPDTB1	3087
READ	3088	READ1	308E	READ2	3095	READ3	3098	READ4	30AD
READ5	30B9	READ6	30BA	READ7	30C5	READ8	30E1	READ9	30E8
READ10	30EC	READ11	30FD	READ12	310F	REM	3110	RETURN	3111
RETUR1	311D	RETUR2	312C	RETUR3	312F	RETUR4	3132	RETUR5	313C
RETUR6	3150	BREAK	3156	STOP	315B	STOP1	3164	STOP2	3183
STOP3	319B	SYNERR	319E	ERROR	31A1	WHAT	31C5	BADINP	31D1
EVAL	31E0	EVAL1	3204	PRCEXP	320B	PRCEX1	3222	PRCEX2	325F
PRCEX3	3267	PRCEX4	326C	PRUNOP	3270	PRUNO1	328F	PRUNO2	3293
PRUNO3	329B	PRUNO4	32AF	PROPND	32B0	PRFN	32BF	PRFN1	32C4
PRFN2	32C5	PRVAR	32C9	PRBRKT	32E4	PRCON	3303	PFUNOP	3308
PFUNO1	331A	PFNEG	3324	PFNOT	3330	PFBIOP	3335	PFADD	3352
PFSUBT	3356	PFMULT	335E	PFDIV	3366	PFAND	336E	PFOR	3376
PFEQL	337E	PFLT	3386	PFGT	338E	PRBIOP	3392	PRBI01	33D7
PRBI02	33E4	PRBI03	33F6	PRBI04	3402	NEGATE	3405	NEGAT1	3415
NEGAT2	3422	ADD	3423	ADD1	343A	SUBT	343E	MULT	3451
MULT1	3466	MULT2	3480	MULT3	3485	MULT4	3490	DIVIDE	3494
DIVID1	34A2	DIVID2	34B5	DIVID3	34C5	DIVID4	34C8	DIVID5	34F6
AND	34FA	OR	3508	NOT	3516	EQUALS	351F	EQUAL1	3523
EQUAL2	3526	LT	352A	GT	3530	COMPAR	3536	COMPAL	3548
PEEK	355A	PEEK1	356A	FINDLN	3580	FINDL1	358B	FINDL2	359D
FINDL3	35A2	FINDL4	35A3	NOTFND	35BC	FOUND	35BD	SEARCH	35C1
GETVAR	35C9	GETVAL	35CF	GETVA1	35D4	NUMBCD	35E1	NUMBC1	35EC
NUMBC2	35F5	ENDNUM	360E	BCDNUM	360F	BCDNU1	361B	HEXASC	3644
HEXAS1	3664	HEXAS2	3674	HEXAS3	367D	ASCHEX	36B2	ASCHX1	36C0
ASCHX2	36C4	ASCHX3	36C8	ASCHX4	36D9	ASCHX5	36DC	ASCHX6	36E2
ASCHX7	3704	ASCHX8	370B	ASCHX9	371C	ADDOP	371D	ADDOP1	373C
ADDOP2	3742	CLASS	3746	CLASS1	3752	CLASS2	375B	CLASS3	375C
IXADD	375D	IXADD1	3764	IXSUB	3765	IXSUB1	376C	FNDSPC	376D
SKPSPC	377A	SKPSP1	3783	CHKSTK	3784	STAKOK	3795	CHKCR1	3799
CHKCR2	37A3	CHKCR3	37AD	BRKISR	37AE	STRTUP	37B4	MESG1	37C2
MESG2	773A	MESG3	37C7	MESG4	37D2	MESG5	37E0	MESG6	76CA
DECTBL	37EE								

