NICK HAMPSHIRE

WITH RICHARD FRANKLIN AND CARL GRAHAM

ADVANCED COMMODORE 64 BASIC REVEALED

Advanced Commodore 64 BASIC Revealed

Also by Nick Hampshire

The Commodore 64 ROMs Revealed
0 00 383087 X

Advanced Commodore 64 Graphics and Sound
0 00 383089 6

The Commodore 64 Kernal and Hardware Revealed
0 00 383090 X

The Commodore 64 Disk Drive Revealed
0 00 383091 8

Advanced Commodore 64 BASIC Revealed

Nick Hampshire

with Richard Franklin and Carl Graham

COLLINS
8 Grafton Street, London W1

Collins Professional and Technical Books
William Collins Sons & Co. Ltd
8 Grafton Street, London W1X 3LA

First published in Great Britain by
Collins Professional and Technical Books 1985

Distributed in the United States of America
by Sheridan House, Inc.

British Library Cataloguing in Publication Data
Hampshire, Nick
Advanced Commodore 64 BASIC Revealed
1. Commodore 64 (Computer)—Programming
2. Basic (Computer program language)
I. Title II. Franklin, Richard III. Graham, Carl
001.64'24 QA76.8.C64

ISBN 0-00-383088-8

Typeset by V & M Graphics Ltd, Aylesbury, Bucks
Printed and bound in Great Britain by
Mackays of Chatham, Kent

Contents

Preface

Whether you program the CBM 64 in BASIC or machine code, an understanding of how the BASIC interpreter works is incalculable in any advanced programming. This book delves into the way the interpreter works and should be used in conjunction with Volume 1 of this series, *The Commodore 64 ROMs Revealed*, when using the interpreter routines within a machine code program.

Knowing how the interpreter operates enables one to perform many interesting functions, probably the most exciting of which is the extension of BASIC with the addition of extra commands, keywords and functions. This book shows exactly how to extend BASIC and includes a package of machine code routines which add over thirty extra commands and functions which enormously improve the power of BASIC. It should be noted that a further set of extended BASIC commands for graphics and sound are contained in Volume 3, *Advanced Commodore 64 Graphics and Sound.*

This book is the product of many years working on Commodore machines, and I am confident that it provides the most complete, interesting and useful information available from any source. All serious programmers should find this an invaluable and constant reference book.

Nick Hampshire

Chapter One

Memory Utilisation by BASIC

1.1 Memory usage

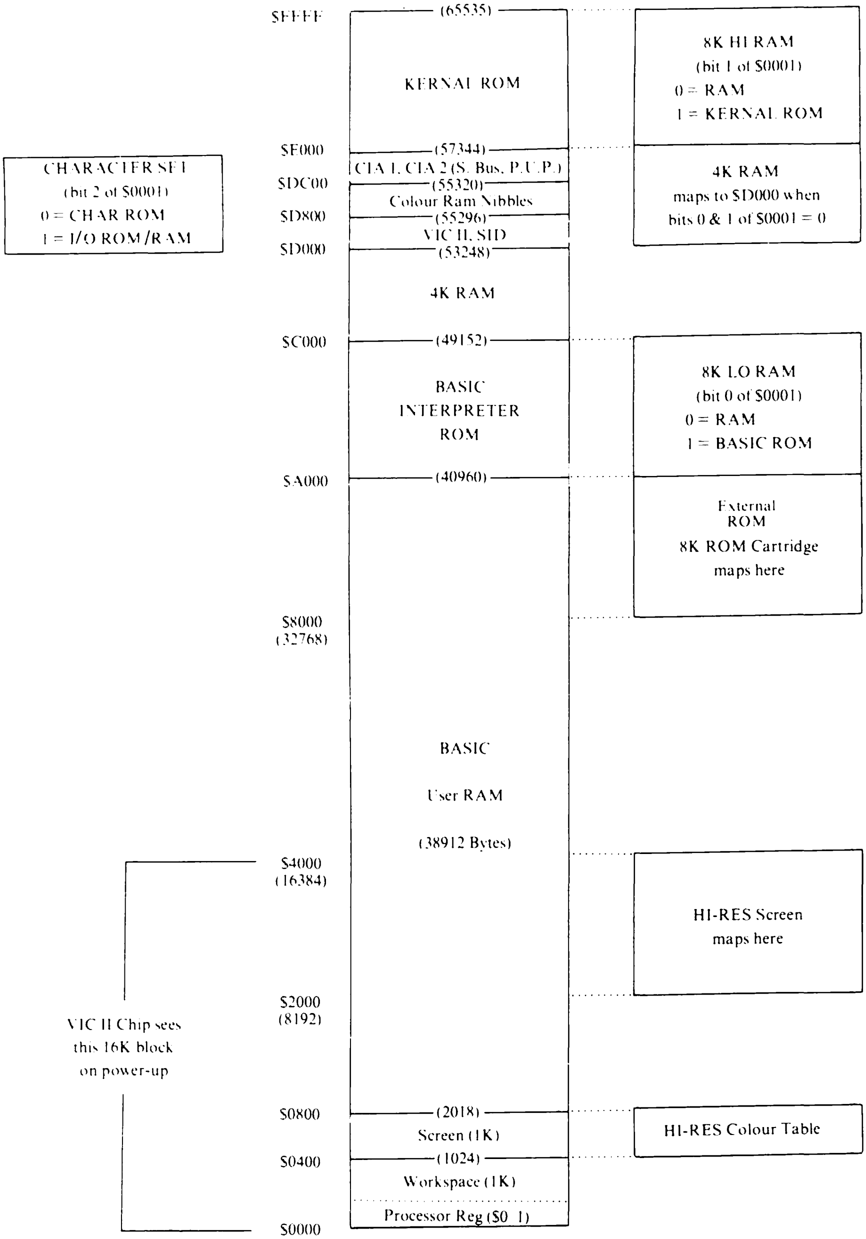


Fig. 1.1. Commodore 64 memory architecture map.

The 6510 microprocessor used in the CBM 64 is capable of addressing up to 65536 bytes of memory. The 64 actually has more memory than this – a total of nearly 88K – and this is accessible by a method known as bank switching. The 64K addressable area of memory is divided into blocks, each having its own function. In the normal memory configuration all 64K of available memory space is allocated to ROM (containing the system software) or RAM memory for storage of variables and programs, and I/O to control the system input and output devices. This division of memory space into blocks is shown in Fig. 1.1, and an understanding of the function of each block is essential if full use is to be made of the CBM 64. The following is a description of each of the memory divisions.

(1) Processor register – hex $00, $01 – decimal 0,1. These two memory locations are the two I/O port control registers on the 6510 microprocessor chip. Address 0 is the data direction port. Any bit set to one will define its corresponding I/O line as an output; a zero will define it as an input. The normal value of the data direction port is binary 00101111 (three input lines and five output lines). The data direction port should, in general, never be changed since these I/O lines are used to define the 64 system architecture. Location 1 is the associated input/output port. In the normal system configuration this location contains, in binary, 00110111. The function of each of these lines is as follows:

Line	Input/output	State	Function
0	output	high	LORAM (0=switch Basic ROM out)
1	output	high	HIRAM (0=switch kernal ROM out)
2	output	high	CHAREN (0=switch character ROM in)
3	output	low	cassette write line
4	input	high	cassette sense switch (0=switch down)
5	output	high	cassette motor control (0=on, 1=off)
6	input	low	undefined
7	input	low	undefined

(2) System variable workspace – hex $0002 to $03FF – decimal 2 to 1023. This area of RAM memory is used to store system variables, buffers, jump vectors and the processor stack. The contents of this area are shown in Table 1.1 on pages 24–28. This table shows not only the location, power up contents and function of each location but also shows how putting different values into certain locations can be used to obtain a range of different effects.

(3) Screen RAM – hex $0400 to $07FF – decimal 1024 to 2047. This area is used to store the ASCII character codes of the characters displayed on the screen. Each memory location corresponds to a character location on the screen. The locations $07F8 to $07FF (decimal 2040 to 2047) are used as the sprite definition pointers.

(4) User RAM area - hex $0800 to $9FFF - decimal 2048 to 40959. This area of memory is used to store programs, data, etc.

(5) Basic interpreter ROM - hex $A000 to $BFFF - decimal 40960 to 49151. The interpreter translates the high level Basic program, step by step, into a series of machine code routines, performing the functions required to execute each command. These routines can be used by other machine code programs; this is dealt with in Chapter 3. A complete annotated listing of the Basic interpreter is contained in *The Commodore 64 ROMs Revealed* in this series.

(6) Free machine code programming RAM - hex $C000 to $CFFF - decimal 49152 to 53247. A 4K block of memory which is not used by Basic and therefore is safe from use by Basic variables and can thus be used to store machine code programs or data.

(7) Video interface controller chip - hex $D000 to $D3FF - decimal 53248 to 54271. This chip uses the first 47 locations of this 1K block (all other locations are unusable). The VIC chip controls the video display, utilising the screen RAM and colour nibble RAM. A full explanation of the function and operation of this chip is given in *The Commodore 64 Kernal and Hardware Revealed* in this series.

(8) Sound interface device - hex $D400 to $D7FF - decimal 54272 to 55295. This uses the first 29 locations of this 1K block (all other locations are unusable). The SID chip controls the sound generation of the Commodore 64. A full explanation of the operation of this chip is given in *The Commodore 64 Kernal and Hardware Revealed.*

(9) Colour nibble memory - hex $D800 to $DBFF - decimal 55296 to 56319. This 1K block of memory parallels the screen memory and is used to store the character colour. It should be noted that this area of memory is only 4 bits wide (normal memory is 8 bits wide).

(10) Complex interface adaptor chip #1 - hex $DC00 to $DDFF - decimal 56320 to 56831. There are two of these I/O devices. The first is the keyboard controller device and is connected to the IRQ line; the second controls the serial I/O ports and provides for the user port. It is connected to the NMI line. Further detailed information on these devices is given in *The Commodore 64 Kernal and Hardware Revealed.*

(11) Basic ROM extension - hex $E000 to $E4FF - decimal 57344 to 58623. This area contains the last section of the Basic interpreter software.

(12) Kernal ROM - hex $E500 to $FFFF - decimal 58624 to 65535. The operating system controls the functioning of the Commodore 64 system, such as initialisation on power up, communications with peripheral devices, screen display and editing, etc. The operating system normally works in conjunction with the Basic interpreter, but the routines within it can be used by any machine code program requiring the operating system functions. A complete annotated listing of the kernal is given in *The Commodore 64 ROMs Revealed* in this

series, and further information on using these routines is given in *The Commodore 64 Kernal and Hardware Revealed.*

1.2 Program storage

1.2.1 The input of a program line

When a program line is entered on the keyboard it is first written into the keyboard buffer. The keyboard buffer is a ten byte block of memory which is used to store keyboard entries temporarily as a first in first out buffer; this is necessary to ensure that no keyboard entries are lost as a result of the system being busy. The operating system routine which enters characters into the keyboard buffer is located at $EA87 and is called by the 60 cycle per second keyboard scanning interrupt.

The keyboard scanning routine takes any keypress, converts it to the correct ASCII code and stores it in the keyboard buffer. If the keyboard buffer is filled, then any further keypresses are ignored until characters are removed by the routine at $E5CD. The routine to remove characters from the keyboard buffer is called either by one of the routines requesting an input from keyboard or by the main warm start routine via the line input routine. The routine to remove characters from the keyboard buffer first blinks the cursor, then removes a character (if there are any) from the keyboard buffer, and in so doing moves all characters in the keyboard buffer down. It then checks that the key pressed was neither the RUN/STOP key nor the RETURN key; if it is neither of these then the character is displayed on the screen. This process is continued until a RETURN key is found, whereupon the line displayed upon the screen is copied into the Basic input buffer.

The Basic input buffer is a block of memory 88 bytes long which is used to store a Basic line when first input, whether it is a program line or a direct mode command. When the warm start routine finds an entry in this buffer with its associated pointers it checks whether the first character in the buffer is a numeric character. If it is numeric then the line is crunched (this converts any Basic keywords into tokens) and then either a line insert or line delete is performed. If the first character is not numeric then the line is crunched and executed, the control jumps to the error checking routine and READY is printed on the screen. After completing either operation the warm start routine returns to get another input.

A flow diagram of the complete character and program line input procedure is shown in Fig. 1.2 on pages 5 and 6.

1.2.2 The tokenised BASIC command

The program line stored in the Basic input buffer is compressed and formatted by the crunch routine. The compression converts each variable length Basic keyword command into a single byte token. The purpose of this is principally to reduce the amount of memory required to store a program, therefore allowing longer programs to be run. Each program line is thus stored in a specific format using the compressed Basic commands. Hence the command PRINT, instead of being stored as five ASCII characters, is stored in a single byte as the decimal value 152. When a program is listed the text compression process is reversed; as far as the user is concerned the program is stored in the same form as it was written.

The following is an example of a tokenised Basic line:

Input	IF	INT	(	A	)	>	5	THEN	PRINT	TAB(	X	)
Tokenised	8B	B5	28	41	29	3E	35	A7	99	A3	58	29

One useful result of text compression, which is well known to most programmers, is the shorthand way of writing Basic commands either in program or direct mode. The rule is that any character in the keyword, except the first or last, can be shifted to terminate that word. A table of all the Basic keywords and their associated tokens is given in Table 1.2 on pages 28–29.

The token value given to a Basic command is a pointer to a table of reserved command words located between $A09E and $A19D and a table of start addresses of Basic commands located at $A00C to $A080. By subtracting 127 from the token value the number of the word in that table can be obtained. The table of reserved commands is used when the commands are crunched. The crunching routine simply scans down the table looking for a word match and counts the number of words tested to obtain the token value. The list routine does the reverse; it scans past the number of words in the table indicated by the token value, then copies the command word into the buffer. The table of execution addresses is used when the command is executed to provide the jump address to the routine which performs that command.

The storage of a program as tokens means that the actual keywords used in a program can be changed without affecting the program's execution. The keywords can be changed by copying the Basic interpreter into RAM and then changing the keyword tables at the start of the interpreter. This will not alter the operation of a Basic program; it will simply change the way it is entered and listed.

1.2.3 Program storage format

Having converted the Basic command into a single byte token the program line is stored together with the line number and a link address at a location just above that of the last line entered. Assuming it is the first line of the program being entered, it will be entered into the following memory locations using this format:

$0800–2048 – contents 0
$0801–2049 – link address lsb (points to starting location of next line)
$0802–2050 – link address msb
$0803–2051 – line number lsb
$0804–2052 – line number msb

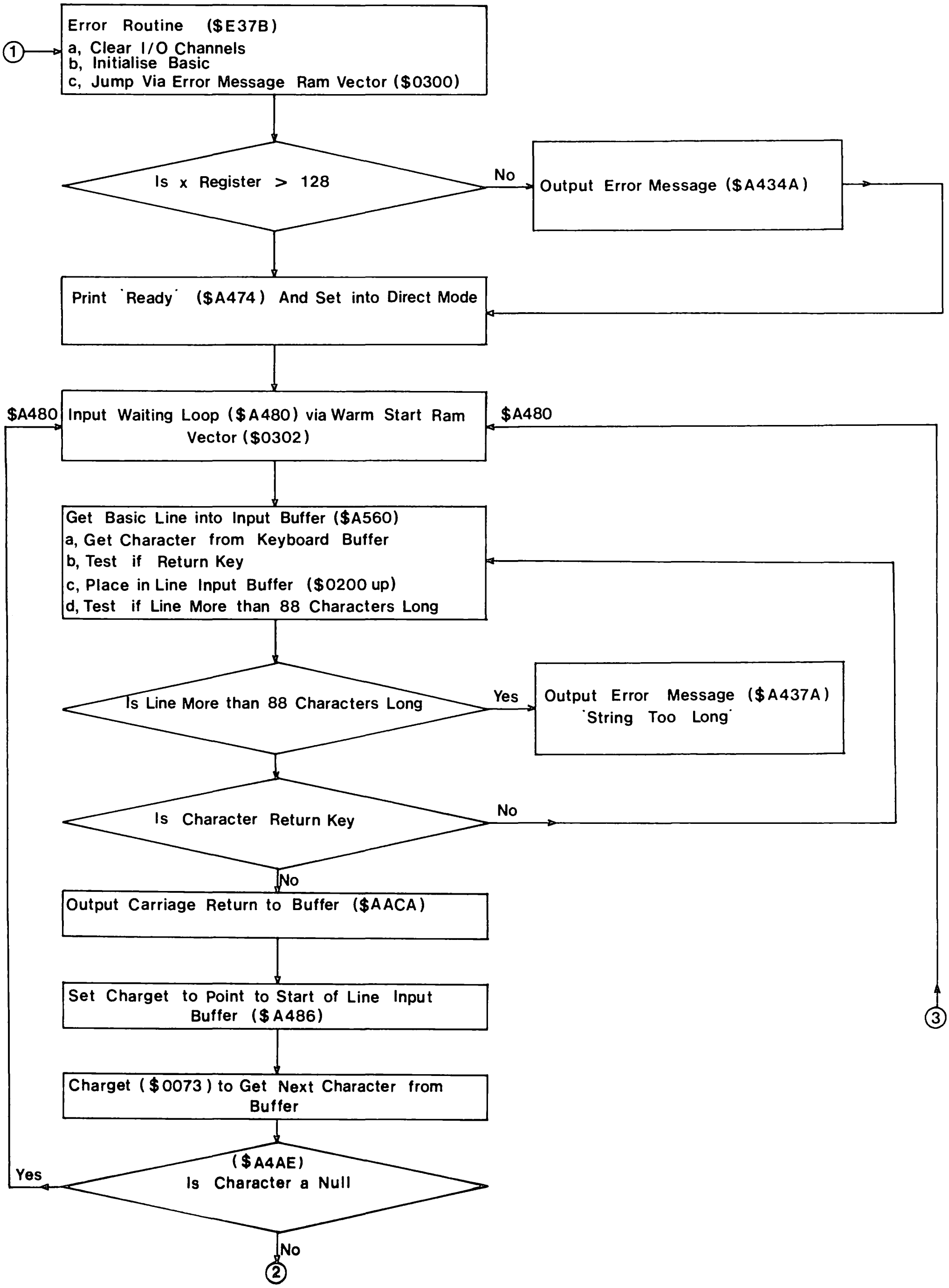


Fig. 1.2. Flow diagram of interpreter BASIC line input.

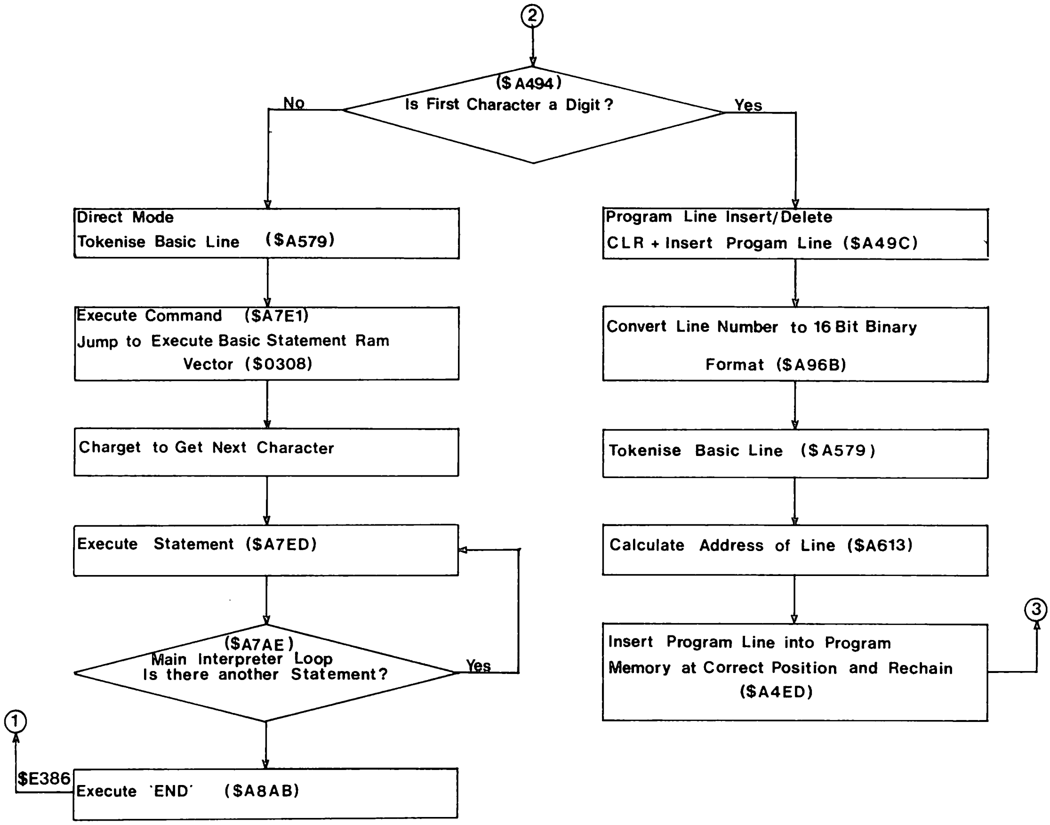


Fig. 1.2. cont.

$0805-2053 – start of compressed Basic text; the number of bytes occupied is variable
$08xx-2xxx – end of line flagged by 0

A Basic program is stored as a series of blocks, each of variable length and representing one line in the program. Each block has a fixed format and all blocks are connected via a link in a sequential list structure. Each line in a program is stored in memory in the correct position dictated by the magnitude of its line number, thus it will be the line with the lowest line number which is stored at the bottom of memory – 2049 up. When a new line is added to a program it is inserted at the correct position and all lines above it are moved up in memory by the size of the inserted line and the links reconnected. The line number is stored in bytes 3 and 4 of a block in a 16 bit binary format. When a program is being run, the current line number being executed is stored in locations $39,$3A. A direct mode of operation is indicated when the contents of location $3A contain the value $FF. The last byte of every block of program line is flagged by a byte, the contents of which is zero. The structure of a program is shown in Fig. 1.3.

The double byte link address which points to the start of the next program

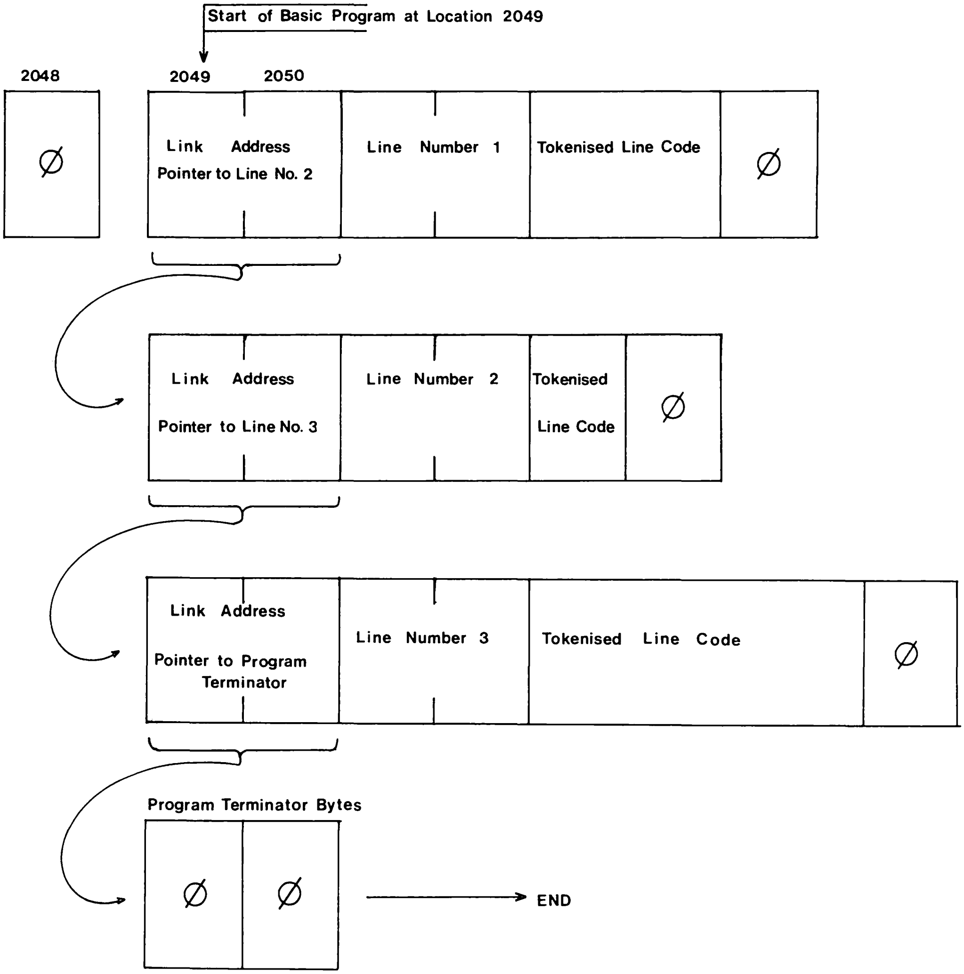


Fig. 1.3. Program line linkage.

line is stored as the first two bytes of a program line block. When lines are added or deleted these link addresses are all recalculated in a re-chaining process. The link address of the last line of the program points not to another line but to two bytes, the contents of which are zero. This forms a link address to zero. It should be noted that when a NEW command is executed it does not erase the whole contents of memory but simply sets the link address of the first line to zero. By reconstructing this link address the program can be restored after a NEW command. Changing link addresses can also be used to hide program lines as part of a security technique or to store machine code subroutines or data.

1.2.4 Using a knowledge of program storage

Having a knowledge of the way a program is stored allows one to perform modifications to program lines. This can be useful in many ways. Examples of the kind of application are line renumbering, an auto line number generator, program compactors, and many others. A line renumbering routine is quite simple, entailing finding the line number of each program line and changing it to the new number. An example of a Basic program to do this is shown in Program 1. This program is designed to be appended to the top of a program and can then be run with a RUN 61000. It should be noted, however, that although this routine will renumber the line numbers it will not renumber jump and branch addresses. For a full renumber routine in machine code which renumbers everything see Chapter 5.

```
61000 REM ** LINE RENUMBER **
61010 INPUT"START LINE,END LINE ,INC";S,E,I
61020 INPUT"NEW START LINE";N
61030 C=256
61040 L=PEEK(43)+PEEK(44)*C
61050 A=L
61060 H=PEEK(A+2)+PEEK(A+3)*C
61070 L=PEEK(A)+PEEK(A+1)*C
61080 IFL=0 THENEND
61090 IFH<S THEN61050
61100 IFH>E OR H>=32767 THENEND
61110 POKEA+2,N AND 255
61120 POKEA+3,N/256
61130 N=N+I
61150 GOTO61050
```

Program 1.

An auto line numbering program is shown in Program 2. Like renumber, this is intended to be appended at the top of a program, and should be entered before starting to write the program. When this program is run it prints the line number on the screen. You then type in the desired program line, and on pressing return the program forces the input line into the low line number part of the program before placing a new line number on the screen. The program line is entered into the program using a special technique which allows program lines to be entered from within a running Basic program. This technique uses the keyboard buffer into which two carriage return characters are poked, the buffer length pointer being set to two characters. The first of these carriage returns fools the interpreter into accepting the line, as entered, and inserts it into the program, taking care of the correct chaining and tokenisation. The second carriage return performs a forced jump and auto run in direct mode to restart the auto line numbering program with the GOTO61040 printed on the screen following the entered line. A much more efficient auto line numbering routine written in machine code is given in Chapter 5 as part of the Basic aid package.

```
61000 REM ** AUTO LINE NUMBER **
61010 INPUT"STARTING LINE NUMBER, INCREMENT";B,L
61020 PRINT"▒▒▒"
61030 POKE830,L:GOTO61060
61040 B=PEEK(828)*256+PEEK(829)
61050 L=PEEK(830)
61055 PRINT".]                    "
61060 PRINT"                                                      .T]"
61070 PRINTB;
61080 OPEN1,0:INPUT#1,A$:PRINT:CLOSE1
61090 PRINT"GOTO61040.TTT]";
61100 POKE198,2:POKE631,13:POKE632,13
61110 B=B+L
61120 POKE828,INT(B/256)
61130 POKE829,B-INT(B/256)*256:END
```

Program 2.

The program compactor in Program 3 uses several different techniques to remove all REM statements. This will speed up both execution time and tape loading time as well as reducing the amount of memory required. As with the preceding two routines it is designed to be appended temporarily to the top of the program. It locates lines containing a REM command token and then removes all following characters by first replacing them with space characters. The line is then displayed on the screen and the terminating spaces removed by using the auto line entry procedure used in Program 2. This procedure removes the spaces and re-chains the program, having moved it all down in memory. Chapter 5 contains a full program compactor routine in machine code which is much more efficient.

```
61000 REM *** REM REMOVER ***
61010 L=PEEK(43)+PEEK(44)*256
61020 GOTO61040
61030 L=PEEK(828)+PEEK(829)*256
61040 N=PEEK(L)+PEEK(L+1)*256
61050 IFL=0THENEND
61060 L=L+4:P=L
61070 Q=PEEK(L)
61080 IFQ=0THENL=N:GOTO61040
61090 IFQ=34THEN62000
61100 IFQ=143THEN62500
61110 L=L+1
61120 GOTO61070
62000 L=L+1
62010 Q=PEEK(L)
62020 IFQ=0THENL=N:GOTO61040
62030 IFQ=34THEN61110
62040 GOTO62000
62500 IFL=PTHENPOKEL,58:GOTO62540
62510 IFL=P+1THEN62530
62520 IFPEEK(L-1)=58THENPOKEL-1,32
62530 POKEL,32
62540 L=L+1
62550 IFPEEK(L)=0THEN62570
62560 GOTO62530
62570 PRINT"▒▒▒LIST";PEEK(P-2)+PEEK(P-1)*256
62580 PRINT"▒                                        ";
62590 PRINT"                                         "
62600 PRINT"▒▒GOTO 61030▒";
62610 POKE828,P-4 AND255
62620 POKE829,(P-4)/256
62630 POKE198,3:POKE631,13:POKE632,13:POKE633,13
62640 END
```

Program 3.

The above are just three of the many possible ways in which an understanding of the way a program is stored can be useful.

1.3 Data storage

The entire area of memory between \$0800 and \$A000 not used for program storage is available for data storage. In addition data can be stored within a program either as DATA statements or defined variables, or directly poked into the 4K block of memory from \$C000 to \$CFFF.

1.3.1 DATA statements

The simplest form of data storage is using data statements. The data in a data statement is stored as ASCII characters on a data statement line within a program. The data is accessed by the program using the READ command. However, data storage in data statements can be added to or changed only by adding or amending program lines in the direct mode. Though the routine in Program 4 can be used to add DATA statements to a program while it is running, this is done by printing the line number followed by DATA and the string or value on the screen, and using the keyboard buffer to force a carriage return and thereby add the line to the program. It should be noted that this procedure will delete all variables and pointers currently used by the program. Another limitation is that data can be accessed from data statements only in a serial mode. This means that to find one particular item the whole table of data must be read. The pointer to the current data statement is stored in locations \$41,\$42 and the data statement line number is stored in locations \$3F,\$40. Manipulation of the contents of these locations provides a means of overcoming this serial access limitation (see the RESTORE command in Chapter 4).

```
61000 REM ** DATA INPUTTER **
61010 INPUT"STARTING LINE NUMBER, INCREMENT";B,L
61020 PRINT"▮▮▮"
61030 POKE830,L:GOTO61060
61040 B=PEEK(828)*256+PEEK(829)
61050 L=PEEK(830)
61055 PRINT"⌉                "
61060 PRINT"                                              :T⌉"
61070 PRINTB;"▮DATA";
61080 OPEN1,0:INPUT#1,A$:PRINT:CLOSE1
61090 PRINT"GOTO61040:TTT⌉";
61100 POKE198,2:POKE631,13:POKE632,13
61110 B=B+L
61120 POKE828,INT(B/256)
61130 POKE829,B-INT(B/256)*256:END
```

Program 4.

1.3.2 Types of variables

Data not stored within the program is stored in an area of memory above the Basic text area as variables. Variables can be divided into two groups. Simple variables are of the kind used in the following statement:

LET X=67

where X is a simple variable. Array variables are defined by a DIM statement and contain more than one value. The number of values is determined by the number of elements in the DIM statement. For both groups of variables there are three types of data – real or floating point numbers, integer numbers and character or string variables where words are stored rather than numbers. The interpreter differentiates between different types of variable by testing the character immediately following the variable name. Thus a variable name followed by a '\$' denotes that it is a string variable, a '%' denotes an integer variable, and if neither of these characters is present, then the variable is a floating point value. If the character following the variable type determinating character is a '(' then this denotes that the variable is an array element. Variable names are thus subject to the following rules:

(1) The first character must be alphabetic.

(2) The second character can be either alphabetic or numeric.

(3) Any further alphanumeric characters are valid but are ignored by the interpreter, thus variable name ABCDE is, as far as the processor is concerned, identical to variable name ABXYZ. Variable names have a practical upper limit on size of 80 characters minus the length of the variable plus one. Long variable names are really only of use to aid comprehension of a program, and since they slow down program execution time should be limited in the final running version. It should be noted that the variable name must never be a reserved Basic word or contain within it a reserved Basic word. Thus a variable called PRINT would be invalid, as would a variable called SPRINT; either of these will give a Syntax error.

(4) The next character after the variable name denotes the variable type; '$' = string and '%' = integer; default is floating point.

(5) If the next character is a '(' then this denotes a subscripted array variable.

(6) If the variable is an array variable then the following values denote the position of the variable within the array.

One useful function when writing a Basic program is to be able to display all variables currently being used and their contents. This is performed by the variable dump routine in Chapter 5.

1.3.3 Simple variables

Simple variables of whatever data type are stored immediately above the Basic program storage area, and start at an address pointed to by the contents of locations $2D, $2E. The amount of memory used to store these variables depends on the number of variables used in the program. Each variable occupies seven bytes of memory, and the top of variables storage area where the next variable may be stored is pointed to by the address in locations $2D, $2E.

For all types of simple variables the first two bytes contain the variable name, the high bit of either byte being used to flag the variable type thus giving four variable types. Examples are:

Variable name	Variable name storage		Variable type
AA	65	65	floating point
AA%	193	193	integer
AA$	65	193	string
FN(AA)	193	65	function definition

Of these four types of variable the first two include the variable value within the seven bytes of the variable, and the last two contain pointers to the variable position in memory (and in the case of a function definition also to the definition). When the program is run and a variable name encountered, the table of variables is sequentially searched for the required variable. If the variable is found then its value is retrieved, otherwise it is added to the end of the variable table. Since each variable occupies the same memory space - seven bytes - the scanning of the variable table is done quite rapidly. However, if speed is important it is a good idea to define all variables required by a speed sensitive portion of the program at the beginning. This will set them up at the start of the variable table and therefore speed up access.

The contents and format of the last five bytes of each variable are different for each variable type. These are shown in Fig. 1.4. The format used to store floating point and integer values is covered in detail in Chapter 2. The pointer used by string variables is a 16 bit address of the start of the string in memory. This can be any RAM memory location, either within the Basic program or from the area of memory at the top of RAM where Basic stores all calculated strings. Byte three of the string variable contains the length of the string. The string, when accessed, is thus fetched from the string pointer location up for the number of bytes indicated by the string length. The format of a function definition variable is different in that it contains two pointers. The first pointer is to the actual definition which is contained within the Basic program. It points to the character following the equals sign in the definition. The second pointer is to the variable used in the definition. This points to the exponent of the variable which is stored in normal floating point format.

Basic will allow variables to be retained when one program is loaded from another, provided the second program is shorter than the first. This could create problems since some strings and all functions have pointers to data within the Basic program. With a new program these pointers will no longer point to correct values and will therefore in most cases give rise to a Syntax error message.

1.3.4 Array variables

The storage of array variables is considerably more complex than that of simple variables. Arrays are stored immediately above the top of the simple variables storage area and their beginning is pointed to by locations \$2F,\$30. The end of the array storage area is pointed to by locations \$31,\$32. It should be noted that adding an extra variable to the simple variable table necessitates moving all array variables up seven bytes in memory, a process which considerably slows down program execution time, and is another reason why it is desirable always to define all simple variables at the start of a program.

Unlike simple variables the three different types of array variables (floating point, integer and string) all use different amounts of memory. However, their general organisation is very similar. All arrays consist of a header followed by a string of variables. The first two bytes of the header contain the array name and use the same convention as simple variables to determine the array variable type (i.e. a setting of bit seven of either character). This is followed by a two byte

Integer Variables

Byte 1	2	3	4	5	6	7
First Characters of Variable Name ASCII +128	Second ASCII +128 or 128	MSB Integer Value in 16Bit 2's Complement Signed Binary	LSB	0	0	0

String Variables

Byte 1	2	3	4	5	6	7
First Characters of Variable Name ASCII	Second ASCII 128 or 128	String Length	LSB 16 Bit Address where String Start is Stored in Memory	MSB	0	0

Floating Point Variables

Byte 1	2	3	4	5	6	7
First Character of Variable Name ASCII	Second ASCII or 0	Exponent in Signed 2's Complement	4 Bit Mantissa Bit Seven contains Mantissa Sign			

Function Definition

Byte 1	2	3	4	5	6	7
First Characters of Variable Name ASCII +128	Second ASCII or 0	LSB 16 Bit Address Pointer to Definition (Byte after = Sign)	MSB	LSB 16 Bit Address Pointer to Variable used in Definition (Points to the Exponent of a simple Floating Point Variable)	MSB	Unused except when Definition Set Up Variable Contents

Fig. 1.4. Storage of BASIC simple variables in memory.

length of the full array entry. The next byte contains the number of dimensions in the array. This is followed by a number of two byte values each containing the value of the dimension, starting with the dimension number mentioned in byte five of the header and descending to dimension zero. This header is then followed by the data. An example of a typical header is as follows:

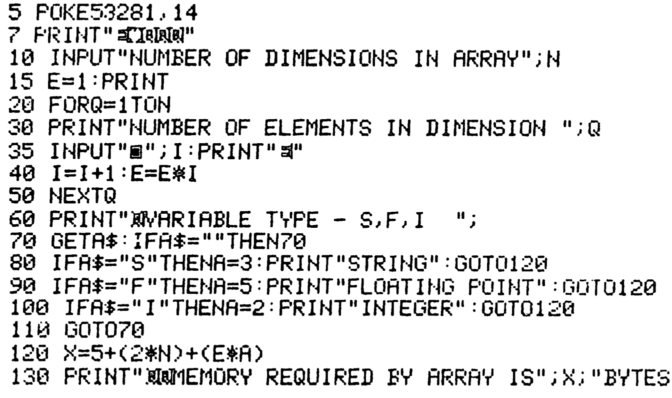
Array AB (5,10,2)

Bytes #	Contents	Function
1	65	first byte of array name (ASCII A)
2	66	second byte of array name (ASCII B)
3	233	high byte of array length
4	3	high byte of array length (1001 bytes)
5	3	number of dimensions
6	0	high byte of number of elements in dimension 2
7	3	low byte of number of elements in dimension 2
8	0	high byte of number of elements in dimension 1
9	11	low byte of number of elements in dimension 1
10	0	high byte of number of elements in dimension 0
11	6	low byte of number of elements in dimension 0
12-1001	x	990 bytes of array data in blocks of 5 bytes (these are all floating point variables)

Note: If the array was an integer array then byte #1 would be 193 and byte #2 194; if a string array then byte #2 would be 194.

Data is stored more efficiently in arrays than in simple variables. Whereas simple variables all occupy seven bytes of memory, array variables occupy five bytes for a floating point variable, two bytes for integers, and three bytes for strings. The format of the number in numerical variables is identical to that of simple variables and is covered in detail in Chapter 4. The three byte string variables consist of a length value in byte one and a two byte pointer to the location of the string in memory. The format of storage of array variables and the array header is shown in Fig. 1.5.

It is quite easy to calculate the amount of memory required by a given array. This is the same value as that stored in bytes three and four of the header. Program 5 can be used:



```
5 POKE53281,14
7 PRINT"▓▓▓▓▓"
10 INPUT"NUMBER OF DIMENSIONS IN ARRAY";N
15 E=1:PRINT
20 FORQ=1TON
30 PRINT"NUMBER OF ELEMENTS IN DIMENSION ";Q
35 INPUT"■";I:PRINT"▓"
40 I=I+1:E=E*I
50 NEXTQ
60 PRINT"▓VARIABLE TYPE - S,F,I  ";
70 GETA$:IFA$=""THEN70
80 IFA$="S"THENA=3:PRINT"STRING":GOTO120
90 IFA$="F"THENA=5:PRINT"FLOATING POINT":GOTO120
100 IFA$="I"THENA=2:PRINT"INTEGER":GOTO120
110 GOTO70
120 X=5+(2*N)+(E*A)
130 PRINT"▓▓MEMORY REQUIRED BY ARRAY IS";X;"BYTES"
```

Program 5.

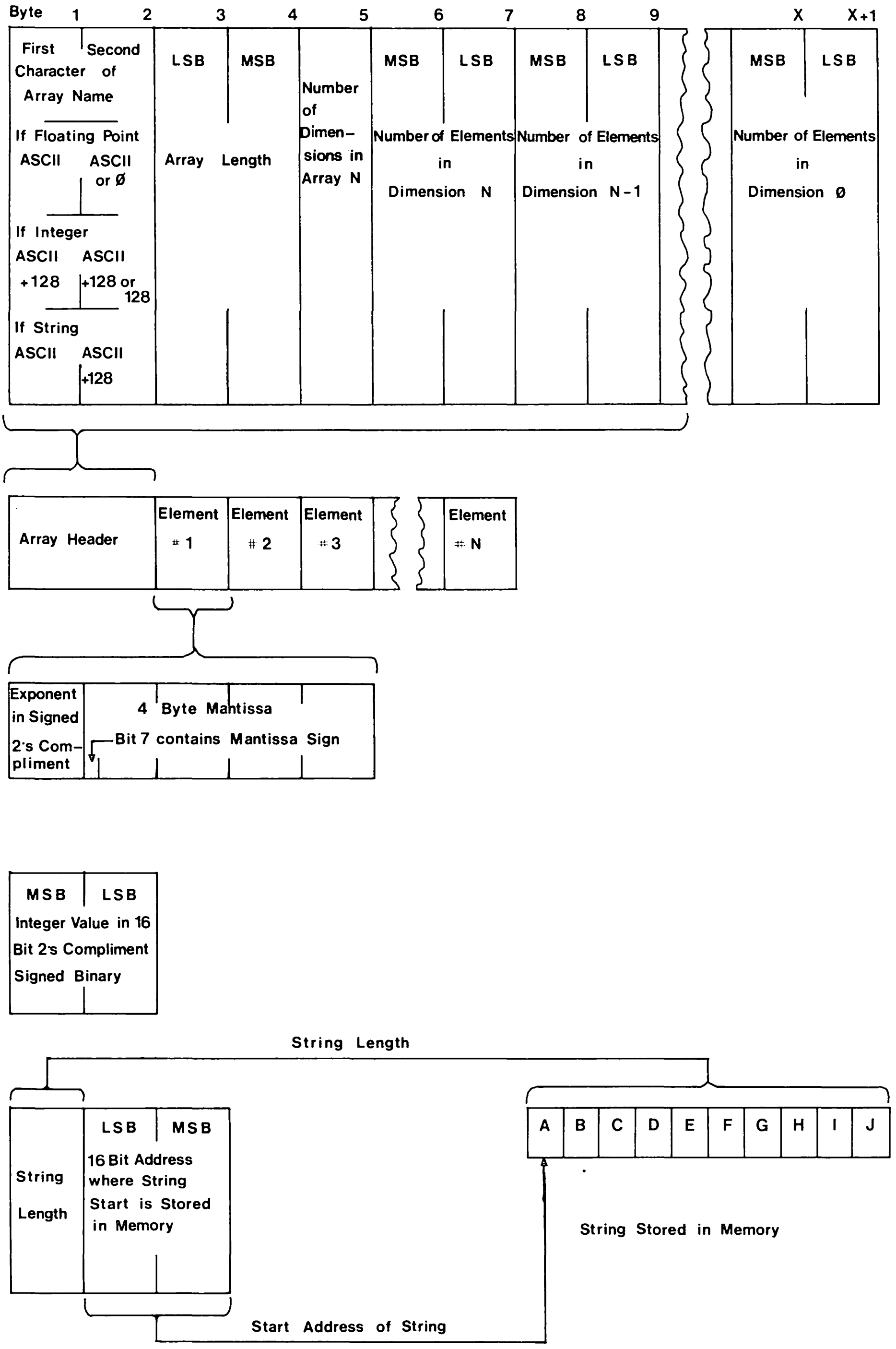


Fig. 1.5. Storage of array variables in memory.

All the variables within an array are held in a strictly defined order. This is best demonstrated in the following example, for an array A(3,2).

Variable #	Array element
1	A(0,0)
2	A(1,0)
3	A(2,0)
4	A(3,0)
5	A(0,1)
6	A(1,1)
7	A(2,1)
8	A(3,1)
9	A(0,2)
10	A(1,2)
11	A(2,2)
12	A(3,2)

As can be seen from this table the first dimension is rotated first, followed by the second, then the third and so on. Thus if there were a third dimension to the above example then element 1 in the third dimension would be accessed between variables 13 and 24. The position of any variable within the array storage area can be determined using the routine in Program 6.

```
10 REM ** ARRAY ELEMENT ADDRESS **
20 POKE53281,14
30 INPUT"██████ARRAY TYPE - S F I ";A$
40 IFA$="S"THENL=3:PRINT"█STRING":GOTO80
50 IFA$="F"THENL=5:PRINT"█FLOATING POINT":GOTO80
60 IFA$="I"THENL=2:PRINT"█INTEGER":GOTO80
70 GOTO30
80 INPUT"█NUMBER OF DIMENSIONS";N
90 DIMDS(N),EN(N)
100 FORI=1TON
110 PRINT"█ NUMBER OF ELEMENTS IN DIMENSION";I
120 INPUTDS(I)
130 IFDS(I)<0THEN110
140 NEXT
150 T=0
160 FORI=1TON
170 PRINT"█ ELEMENT NUMBER ";I
180 INPUTEN(I)
190 IFEN(I)<0OREN(I)>DS(I)THEN170
200 NEXT
210 DS(0)=0
220 FORI=1TON
230 T1=1
240 FORQ=0TOI-1
250 T1=T1*(DS(Q)+1)
260 NEXT
270 T=T+T1*EN(I)
280 NEXT
290 T=T*L+5+N*2
300 PRINT"███ELEMENT OFF SET FROM START OF ARRAY"
310 PRINTT
```

Program 6.

To determine the exact position within memory the value obtained from Program 6 must be added to the memory address of the start of the array. If this is the first array then this address is stored in double byte format in locations $2F,$30. If it is not the first array then the size of all preceding arrays must be calculated and added to the start of array storage address.

1.4 Using BASIC variables within machine code routines

Where machine code subroutines are called from a Basic program it is sometimes useful to pass parameters and data using existing Basic variables. Other machine code routines such as a sort would be specifically designed to manipulate Basic variables and arrays. If simple variables are used to pass parameters or data then they should be set up as the first variables within the variable table. This means that the first program line defines them using dummy values. These variables are easily accessed using the start of variable pointer and adding this to the index to variable pointer multiplied by seven. This will point to the first byte of the variable name which can then be verified and the data utilised using the routines within the interpreter to handle floating point values.

Array data can be accessed using the method employed in Programs 5 and 6. An example of such an application would be using an integer array to store a screen display, using the high byte to store the character and the low byte the colour. Such an array would use no more memory than storing it in memory using poke commands, but would be faster and allow interesting manipulation from Basic. If string arrays are to be sorted then this can be easily achieved by simply swapping the pointers stored in the array (see Chapter 5 for an example of this).

1.5 Interpreter routines to handle variables

The interpreter contains many different routines to handle and manipulate variables; some useful ones are detailed in the rest of this section. Before using any of these or other variable handling routines within your own machine code programs, it is highly advisable to study the documented source code for all these interpreter routines which is contained in *The Commodore 64 ROMs Revealed* in this series. For routines handling variable input/output and manipulation see the relevant keywords in Chapter 3.

1.5.1 Some useful routines

Routine: Search for variable

Entry point: $B08B

Function: The first function of this routine is to validate the variable name. The first character must be alphabetic though the second can be either alpha or numeric. The variable type is also determined and the flag in $0D is set accordingly. If the variable is numeric then $0D=$00 and if it is string =$FF. The numeric type flag in $0E is also set to $00 if it is a floating point and to $80 if it is integer. If the variable name is followed by a left bracket then the routine branches to $B1D1 which finds or makes an array. The variable name is stored in locations $45,$46. Having verified the variable name and determined the type the routine searches for the variable in the section of memory allocated to variable storage. If found then the variable address pointer is returned in $5F and $60. If the variable is not found then the routine branches to $B11D where a new variable is created.

Input parameters:
$45 - first character in variable name
$46 - second character in variable name

Output parameters:
$0D - variable type flag
$0E - numeric type flag
$5F - lsb of address of variable
$60 - msb of address of variable
Note: The values must conform to the variable type flag convention covered earlier in this chapter.

Error messages: Syntax error - if the first character of the variable name is not alphabetic

Example use: To find the location of variable AB$.

```
    LDA #$41     ;ASCII code for first variable name character
    STA $45      ;put in first current variable name store
    LDA #$C2     ;ASCII code for second variable name character
    STA $46      ;put in second current variable name store
    JSR $B08B    ;find variable location
```

Routine: Print string from memory

Entry point: $AB1E

Function: The starting address of the string to be printed is stored in the accumulator (lsb) and .y index register (msb) prior to entering this routine. Consecutive characters are then printed to the current output device until a zero terminator byte is encountered.

Input parameters:
.a - lsb of start address of string
.y - msb of start address of string

Output parameters: None

Error messages: None

Example use: To print a string starting at location $C000 to the current output device.

```
LDA #$00   ;lsb of string start address
LDY #$C0   ;msb of string start address
JSR $AB1E  ;output string
```

Routine: Set up string

Entry point: $B487

Function: This routine creates space at the top of memory for a string, puts it there and sets the pointers. On entry the starting location of the string is stored in .a (lsb) and .y (msb). This starting address could be either the input buffer at $0100, in which case it would have a zero terminating byte, or a string within quotes in a Basic program. The string origin is determined by the flags in locations $07,$08. On exit the string length is stored in $61 and the address pointer in $62 (lsb) and $63 (msb).

Input parameters:
.a lsb of start of string address
.y msb of start of string address
$07,$08 flags for quotes

Output parameters:
$61 string length
$62 string address pointer lsb
$63 string address pointer msb

Error messages: Formula too complex if insufficient stack space

Example use: Get a string from the buffer starting $0100 and put it in the string storage area.

```
LDA #$00 ;lsb of buffer start address
LDY #$01 ;msb of buffer start address
JSR $B487 ;transfer to string storage area
```

Note: The address pointers are returned in $62,$63 and the length in $61 can be inserted into the requisite locations of a string variable located using the routine at $B08B.

Routine: Discard unwanted strings

Entry point: $B6A3

Function: This clears the last entered string pointed to by locations $64,$65, and moves the bottom of the string pointers up by the size of the string length so that a new string will overwrite it. This routine is used to overwrite the last entered string only. On exit locations $22,$23 point to the removed string.

Input parameters:
$64 – lsb of address of last entered string
$65 – msb of address of last entered string

Output parameters:
$22 - lsb of address of removed string
$23 - msb of address of removed string

Error messages: None

1.6 How BASIC works

There are two sides to the functioning of the Basic interpreter; program entry and program execution. Program entry is nearly always performed in direct mode, while program execution is carried out principally in run mode (except for single line program or command execution in the direct mode). Program entry has already been dealt with in the section on program storage.

The entry to the program execution loop is via one of the execution commands entered in direct mode. These commands are RUN, GOTO and GOSUB. When one of these commands is executed in the direct mode it sets the charget pointers to the beginning of the program or the designated line number (the charget subroutine is described in Chapter 4) and then goes to the main program interpreter loop where the rest of the program is executed. For explanations of the functioning of the routines for RUN, GOTO and GOSUB see Chapter 3, and for the source code interpretation of these routines and the program execution routines see *The Commodore 64 ROMs Revealed* in this series.

The program execution loop is fairly straightforward and consists of two quite short routines. The logic flow within these routines is shown in the flow diagram in Fig. 1.6. The two routines are the main Basic interpreter control loop and the execute Basic statement routine. The function of these two routines is as follows.

*1.6.1 Main BASIC interpreter loop - start *A7AE*

This loop routine controls the execution of a Basic program, and has the following sequence of operations:

(1) Check for the STOP key. If pressed, exit loop to direct mode.
(2) Check for the end of line or a program terminator (0 = end of line and 00 = end of program). If it is the end of the program then execute the END routine, otherwise locate next program line.
(3) Put the next character of the Basic line into the accumulator using the charget routine.
(4) Jump to the execute Basic statement routine and then return to the start of the interpreter loop at $A7AE.

1.6.2 Execute BASIC statement routine - start $A7ED

The character obtained by charget in step 3 of the interpreter loop is in the accumulator. This character is first checked to see if it is a line terminating zero. If so then the routine returns to the interpreter control loop at $A7AE and starts

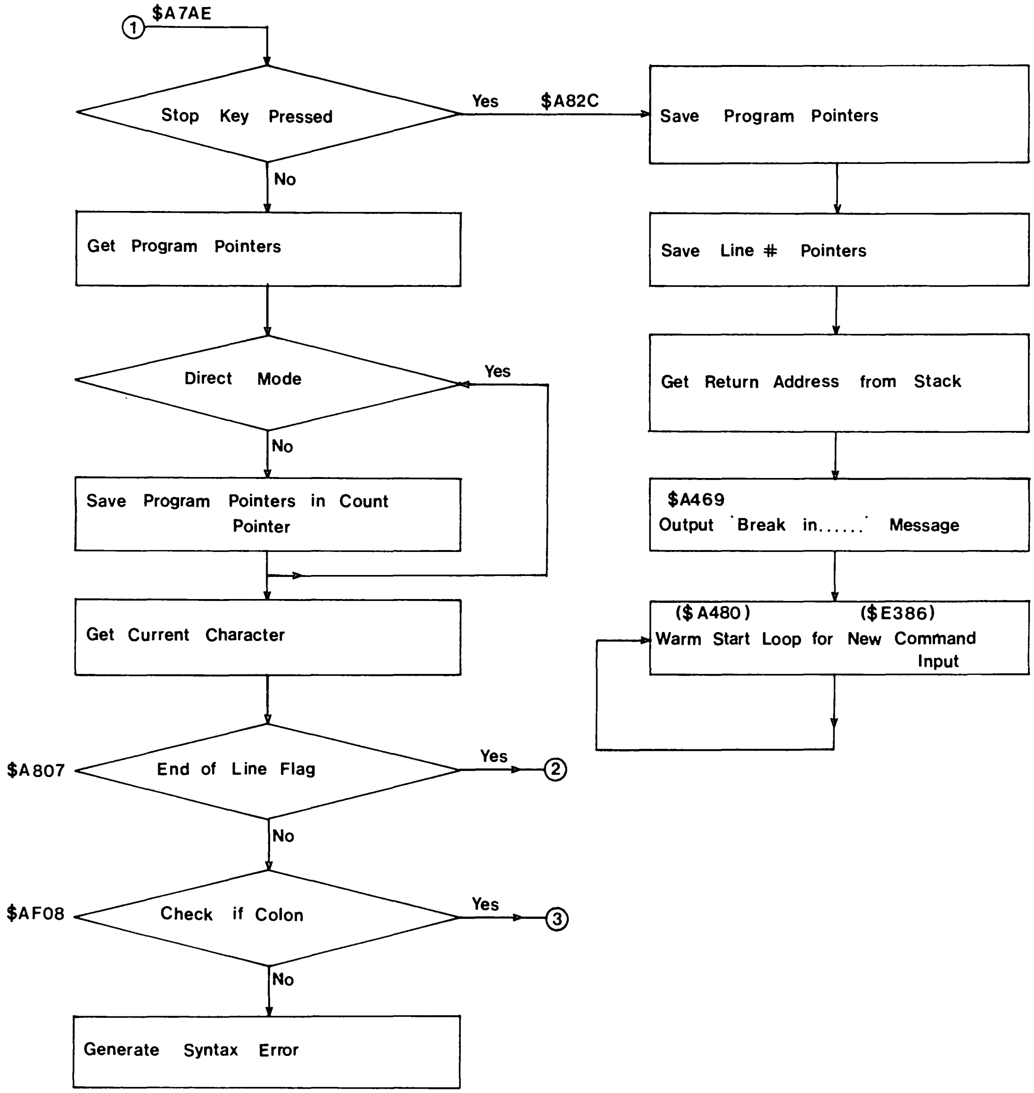


Fig. 1.6. Main BASIC interpreter loop.

on the next line. The character in the accumulator is then checked to see if it is a token (this is assumed if the code value is greater than $80). If a token is not the first character found in a statement then the character is assumed to be a variable and a LET default assignment is performed. When a token is found it is first checked to see if it is a function or the GOTO command; if it is then these statements are performed. The token value is then used as a pointer to the keyword table (starting at $A00C) by subtracting $80 and multiplying the result by two. This pointer is used to get a two byte address of the start of the routine which performs the command. The two byte address is pushed onto the stack

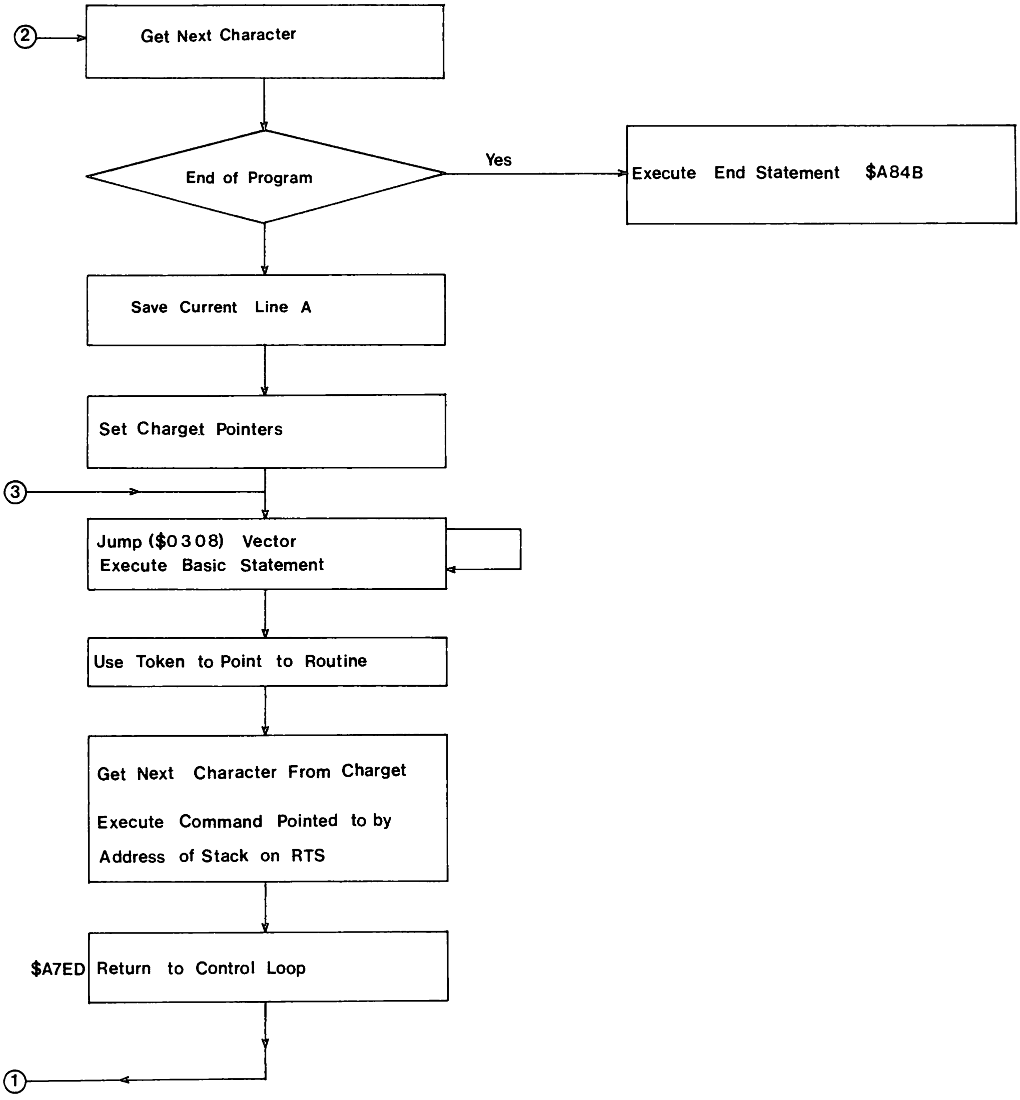


Fig. 1.6. cont.

and a jump to the charget routine performed. Charget puts the next character of the program line into the accumulator (this is usually a parameter required to execute the command), and since charget terminates in an RTS instruction it will return not to the general execute statement routine but to the routine starting at the address stored on stack – the command routine. On returning from the command routine the control will return to the start of the Basic interpreter loop.

The execution of a Basic command is duplicated in the token handling routines for adding commands to Basic; these routines are given in Chapter 4.

Table 1.1. BASIC zero page storage.

In this table are the addresses of zero page storage for the Basic interpreter. These location are from $03-$8F (3-143). Locations 0 and 1 are the processor registers, location 2 is unused and locations above 143 are the kernal storage area.

$03-$04 3-4 *Initial:* Hex $B1AA Dec 45482
This is a two byte vector for the Basic to use to convert numbers in floating point format into two byte signed integers. This vector could be changed to point to your own routine if required (i.e. for rounding up the value). This value remains unchanged.

$05-$06 5-6 *Initial:* Hex $B391 Dec 45969
This is a two byte vector for the Basic to use to convert numbers in two byte signed integer format into floating point. This value remains unchanged.

$07 7 *Initial:* Not applicable
This byte is used in the main interpreter loop to store a character whilst searching for the next Basic statement on a line (or next line). There is no way of manipulating this byte.

$08 8 *Initial:* Not applicable
This byte is used in the Crunch to tokens routine and is used as a flag as to whether the next character is to be crunched or not. This value has no effect unless the characters follow the quotes character, REM, or DATA. It could be possible to wedge into the Crunch to tokens link and put into the correct position a store to location 8 with an illegal value (#$FE). This would then cause the input line not to be crunched.

$09 9 *Initial:* Not applicable
This location stores the position on a line where the next byte is to be displayed. This is only ever used when the TAB command is found in a PRINT command. At this point, the value in this location is subtracted from the TAB value and if greater than zero, that number of cursor movements to the right is printed.

$0A 10 *Initial:* Not applicable
This byte just stores a 1 or 0 to say whether a file is being loaded or verified. The kernal has a byte with the same use.

$0B 11 *Initial:* Not applicable
This location is used as a storage for the position in the input buffer where the Crunch to tokens routine is. Also in the same routine is the token value minus $80. This location is also used to store the number of subscripts of an array when setting up/reading etc.

$0C 12 *Initial:* Not applicable
This value is a flag to tell the Find array routine whether the array exists or not. If not, then the array is created to the default dimension (10) and the number of subscripts (max 3).

$0D 13 *Initial:* Not applicable
This location is a flag set by the find variable routine which just says whether the variable was string ($FF) or numeric ($00).

$0E 14 *Initial:* Not applicable
This location holds the flag, if the variable was numeric, to state whether it was integer ($80) or real ($00).

$0F 15 *Initial:* Not applicable

This byte is used in the LIST routine to say whether a token is to be converted to text or just displayed as the ASCII character. It is used for quotes, REM, and DATA.

$10 16 *Initial:* Not applicable

This location is used by the DEF FN and check FN syntax routines. This byte is also used when searching for or creating a variable.

$11 17 *Initial:* Not applicable

This location is used to flag whether a certain input is from READ ($98), GET ($40) or INPUT ($00).

$12 18 *Initial:* Not applicable

This byte is used as a flag for the TAN command (sign) and the comparison routines (result).

$13 19 *Initial:* Hex $00 Dec 0

Current I/O prompt flag. This byte is checked by the INPUT command to see whether the prompt flag '?' is to be displayed. Setting this value to a one will cause the prompt to be 'turned off'.

$14-$15 20-21 *Initial:* Not applicable

This two byte value is the integer value location. All commands using a two byte integer (signed or unsigned) use this location, an example being the POKE command where the address is stored in these locations.

$16 22 *Initial:* Hex $19 Dec 25

This location is the pointer to the temporary string stack. The temporary string stack is nine bytes long and is used when evaluating an expression.

$17-$18 23-24 *Initial:* Not applicable

This two byte vector is a pointer to the last temporary string used.

$19-$21 25-33 *Initial:* Not applicable

This is the nine byte long temporary string stack. This stack is used by the string manipulation routines before setting the string to point to it.

$22-$25 34-37 *Initial:* Not applicable

These four bytes are used as a temporary pointer area by some of the Basic routines. It is usually safe to use these in your own routines but do not depend on the values remaining after exit from your routine.

$26-$2A 38-42. *Initial:* Not applicable

These five bytes are used to store products from the multiplication routines. The numbers are stored in five byte packed format (as with variables).

$2B-$2C 43-44 *Initial:* Hex $0801 Dec 2049

This vector is the pointer to where the Basic program starts in memory. This value is not changed once the Basic interpreter has been initialised. The value can be changed before loading a program so that some memory below the Basic program is protected. For example: POKE43,1:POKE44,64 will protect the bottom bank from the program. Unfortunately this will reduce the size of the program area by 14K but will allow user defined characters and sprites to be stored without worry of corruption.

Note: Another POKE is required to ensure that the program will RUN: POKE(PEEK (43)+PEEK(44)*256)−1,0.

$2D–$2E 45–46 *Initial:* Hex $0803 Dec 2051

This vector is the pointer to the start of the variable storage area. Its value always points to the location two bytes after the Basic program, thus it is changed every time a program line is changed.

$2F–$30 47–48 *Initial:* Hex $0803 Dec 2051

This vector is the pointer to the end of Basic variable storage. Before a variable is declared this vector is the same as the start of variable storage. Each time a variable is set up, this value will be increased by seven bytes (for simple strings, integer, real variables and functions). This vector is also the pointer to the start of array storage.

$31–$32 49–50 *Initial:* Hex $0803 Dec 2051

This vector is the pointer to the end of array storage. Before any array is declared this value is the same as the start of variable storage. Each time an array is set up, the pointer is increased by the length of the entry. (This value is variable depending on the number of dimensions and the size of each dimension.)

$33–$34 51–52 *Initial:* Hex $A000 Dec 40960

This vector is the pointer to the position where the last string was put. Strings are stored from the top of memory working downwards. When the string pointer passes the end of array pointer a garbage collect is done. This discards all strings that are not pointed to, thus giving as much free memory as possible. If this does not give enough memory to insert a variable, the message Out of memory will be displayed.

$35–$36 53–54 *Initial:* Hex $A000 Dec 40960

This vector is the utility string pointer.

$37–$38 55–56 *Initial:* Hex $A000 Dec 40960

This vector points to the first unusable byte at the top of memory (normally the beginning of the Basic ROM). This value is not changed by the interpreter but can be changed by you to protect an area at the top of the Basic program for the use of machine code routines, data, etc.

$39–$3A 57–58 Initial: Hex $FFxx Dec >65279

This two byte value is the store for the current Basic line number of the line being operated on. The high byte is set to $FF to say that Basic is in direct mode (it disables GET and INPUT).

$3B–$3C 59–60 *Initial:* Not applicable

This two byte value stores the line number of the previous Basic line used.

$3D–$3E 61–62 *Initial:* Hex $FFxx Dec >65279

This vector is the pointer to the Basic statement to be operated on when the command CONT is called. *Note:* Do not use CONT inside a program as this value will point to itself (endless loop).

$3F–$40 63–64 *Initial:* Hex $0000 Dec 0

This two byte value is the line number where the next value for READ is taken from in a DATA statement.

$41–$42 65–66 *Initial:* Hex $0800 Dec 2048

This vector is the pointer to the memory of the first byte of the next DATA value.

$43-$44 67-68 *Initial:* Not applicable
This vector is the pointer to where the input for READ, GET, and INPUT is stored to convert to number form (if need be).

$45-$46 69-70 *Initial:* Not applicable
These two bytes store the name of the last variable accessed. The high bits are set to give the correct type as well.

$47-$48 71-72 *Initial:* Not applicable
This vector is the pointer in memory to the last variable accessed.

$49-$4A 73-74 *Initial:* Not applicable
This vector is the pointer to the variable being used in the current FOR...NEXT loop.

$4B-$4C 75-76 *Initial:* Not applicable
These two bytes are used as a temporary storage for things such as Basic pointers.

$4D 77 *Initial:* Not applicable
This byte is the comparison symbol accumulator which holds which comparison symbols have been found in the Evaluate expression routine.

$4E-$53 78-83 *Initial:* Not applicable
These six bytes are a work area for miscellaneous routines.

$54-$56 84-86 *Initial:* Not applicable
Location $54 holds the byte value for 'JMP' and the other two bytes are set up when a function is encountered.

$57-$60 87-96 *Initial:* Not applicable
These ten bytes are floating point accumulators three and four and are temporary areas for some of the arithmetic routines.

$61-$66 97-102 *Initial:* Not applicable
This is floating point accumulator one. All calculations use these locations and the results of all arithmetic routines are left in here.

$61 - exponent value
$62-$65 - mantissa
$66 - sign

$67 103 *Initial:* Not applicable
Some of the arithmetic routines use one of the two series to perform the calculation. This location holds the number of constants required for the series.

$68 104 *Initial:* Not applicable
This byte holds the overflow from FPACC#1 when some calculations are performed.

$69-$6E 105-110 *Initial:* Not applicable
Floating point accumulator two.

$6F 111 *Initial:* Not applicable
This byte holds the sign comparison byte for FPACC#1 and FPACC#2 for the use of division etc.

$70 112 *Initial:* Not applicable
This byte contains the underflow from FPACC#1. It is used when transferring the value into memory. The byte may also be referred to as the 'rounding' byte.

$71-$72 113-114 *Initial:* Not applicable

The main use of this vector is as the pointer to a series constant.

$73-$8A 115-138 *Initial:* See text

This is the location of the zero page routine used by Basic to get the next character from the current input line (charget). For more information see Chapter 4.

$8B-$8F 139-143 *Initial:* See text

This is the seed value from which the next RND value will be calculated. The initial values are:

$80,$4F,$C7,$52,$58

Table 1.2. Table of BASIC keywords and their tokens.

Token value		
Decimal	Hexadecimal	Keyword
128	$80	END
129	$81	FOR
130	$82	NEXT
131	$83	DATA
132	$84	INPUT#
133	$85	INPUT
134	$86	DIM
135	$87	READ
136	$88	LET
137	$89	GOTO
138	$8A	RUN
139	$8B	IF
140	$8C	RESTORE
141	$8D	GOSUB
142	$8E	RETURN
143	$8F	REM
144	$90	STOP
145	$91	ON
146	$92	WAIT
147	$93	LOAD
148	$94	SAVE
149	$95	VERIFY
150	$96	DEF
151	$97	POKE
152	$98	PRINT#
153	$99	PRINT
154	$9A	CONT
155	$9B	LIST
156	$9C	CLR
157	$9D	CMD
158	$9E	SYS
159	$9F	OPEN

Token value		
Decimal	Hexadecimal	Keyword
160	$A0	CLOSE
161	$A1	GET
162	$A2	NEW
163	$A3	TAB(
164	$A4	TO
165	$A5	FN
166	$A6	SPC(
167	$A7	THEN
168	$A8	NOT
169	$A9	STEP
170	$AA	+
171	$AB	−
172	$AC	*
173	$AD	/
174	$AE	↑
175	$AF	AND
176	$B0	OR
177	$B1	>
178	$B2	=
179	$B3	<
180	$B4	SGN
181	$B5	INT
182	$B6	ABS
183	$B7	USR
184	$B8	FRE
185	$B9	POS
186	$BA	SQR
187	$BB	RND
188	$BC	LOG
189	$BD	EXP
190	$BE	COS
191	$BF	SIN
192	$C0	TAN
193	$C1	ATN
194	$C2	PEEK
195	$C3	LEN
196	$C4	STR$
197	$C5	VAL
198	$C6	ASC
199	$C7	CHR$
200	$C8	LEFT$
201	$C9	RIGHT$
202	$CA	MID$
203	$CB	GO

Chapter Two

Arithmetic Processing by BASIC

2.1 How BASIC stores and uses numbers

2.1.1 Numeric variables, types and range

Basic uses two different types of numbers; integer and floating point. An integer number is stored as two bytes giving a sixteen bit signed number which can store numbers in the range +32767 to −32768. Floating point numbers require five bytes and can store much larger values in the range +−1.70141183 E38 to +−2.93873588 E−39. In the Basic interpreter all calculations, whether on integer or floating point values, are performed using floating point values rather than simple integers or binary values. Consequently all integer values are first converted to floating point format before any calculations are performed.

The format for the storage of an integer value is very simple, consisting of two bytes stored as low order/high order bytes. Negative values are stored in a twos complement form. Floating point values are stored in either packed form occupying five bytes, or unpacked form in six bytes. Packed format is the normal mode for storing floating point variables in memory. Unpacked format is used when performing calculations upon floating point values. In either format there are three components of a floating point value; the sign, the exponent, and a four byte mantissa. In packed mode the sign is stored as bit seven of the most significant byte for the mantissa; in unpacked format the sign occupies its own byte.

2.1.2 The floating point accumulator

In order to perform arithmetic operations on any floating point value the interpreter needs temporary storage locations for the values being worked upon and the result. There are two principal work areas, known as floating point accumulator #1 and floating point accumulator #2. These names are usually shortened to FAC#1 and FAC#2. Each floating accumulator occupies six bytes; FAC#1 starts at $61, and FAC#2 at $69. There are, in addition, three further areas where floating point numbers in packed format (occupying five bytes) are stored; these areas start at $57, $5C and $26. The format and location of the two floating accumulators are as follows:

Location FAC#1	FAC#2	Function
$61	$69	Exponenent + $80
$62	$6A	Mantissa msb
$63	$6B	Mantissa byte #2
$64	$6C	Mantissa byte #3
$65	$6D	Mantissa lsb
$66	$6E	Sign ($FF = – and $00 = +)

Other locations used are:

$68 overflow byte for FAC#1
$6F sign comparison byte
$70 rounding byte for FAC#1

2.1.3 How a floating point number is stored

The storage of a floating point number is fairly complex both in packed and unpacked format. The data used to store a floating point number can be divided into three components; the exponent, the sign, and the mantissa. In the unpacked format the exponent and sign both occupy one byte and the mantissa four bytes. The following is an explanation of each component of a floating point number.

Exponent The exponent indicates the position of the decimal point within the number. Bit seven of the exponent byte indicates the sign of the exponent, thus if the exponent is positive, bit seven is set to one and therefore the value of the exponent byte will always be greater than 128. If the exponent is negative then bit seven is set to zero and the exponent value is less than 128. The exponent is stored as a power of 2 and is multiplied by the mantissa value to produce the final value. The following formula can be used to convert a number N stored in the mantissa bytes (see Mantissa below for calculation of N) to the full floating point number by multiplying it with a positive exponent:

Value = N * 2 ↑ (E–129)

To determine the exponent of a number, find the highest power of two which can be subtracted from the number. Thus if the number is 18.256, then the highest power of two is 16 or 2↑4. The exponent value is positive and therefore equals 129+4 or 133. The fact that the exponent is derived in this way means that the mantissa for two different values may be the same, with the difference being registered solely by the contents of the exponent. Thus the floating point mantissa contents for the values 3.14159 (pi) and 6.28318 (pi*2) are identical:

3.14159 stored as: exponent 130 and mantissa 73,15,218,161
6.28318 stored as: exponent 131 and mantissa 73,15,218,161

As can be seen from this, multiplying and dividing a floating point number by two is a very simple operation involving adding or subtracting one from the exponent. The range of the exponent is +− 2↑128; this equates approximately to +− 10↑38.

Sign The sign of the value is stored in unpacked format as a single byte with a value of $FF for negative numbers and $00 for positive numbers. In packed format the sign is stored in bit seven of the highest byte of the mantissa. If bit seven is zero then the mantissa is positive and if one then it is negative. Thus the packed floating point values for +2 and −2 are:

number +2 is: exponent 130 and mantissa 0,0,0,0
number −2 is: exponent 130 and mantissa 128,0,0,0

Mantissa The mantissa is stored in four bytes minus the most significant bit of the most significant byte of the mantissa which is used to store the sign bit. To convert a number stored in the mantissa into its numeric equivalent use the following formula:

N = 1+((M1 AND 127)+(M2+(M3+M4/256)/256)/256)/128

where M1,M2, M3 and M4 are the mantissa bytes, with M1 the highest and M4 the lowest. When N has been obtained it should be multiplied by 2↑ (exponent −129) to give the actual value. Program 7 allows the input of a number then prints the contents of the exponent and mantissa bytes for that number as it is stored in floating point format. These values are then used by lines 90 to 120 to convert the floating point byte values back into the number.

To convert a number into floating point form is a slightly harder calculation and involves the following steps:

(1) Find the highest power of two which can be subtracted from the number. E = the value of two to this highest power.

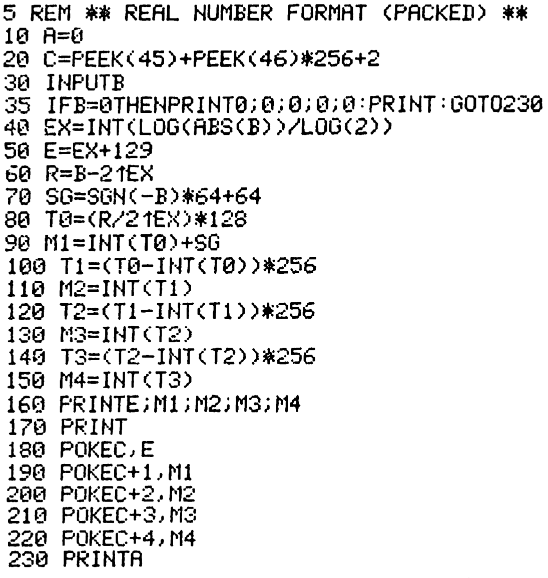
```
5 REM ** REAL NUMBER FORMAT (PACKED) **
10 A=0
20 C=PEEK(45)+PEEK(46)*256+2
30 INPUT" A REAL NUMBER";A
40 E=PEEK(C)
50 M1=PEEK(C+1)
60 M2=PEEK(C+2)
70 M3=PEEK(C+3)
80 M4=PEEK(C+4)
90 PRINT
100 PRINTE;M1;M2;M3;M4
105 IFE=0THENPRINT0:END
110 SG=SGN(64-(M1 AND 128))
120 N=(M1 AND127)+128
130 N=N*256+M2
140 N=N*256+M3
150 N=N*256+M4
160 N=N*2↑(E-160)*SG
200 PRINTN
```

Program 7.

(2) Let R = the remainder after subtracting the value of 2↑E. The calculation is then as follows:

T0 = (R/E)*128
M1 = INT(T0)+mantissa sign (sign = 0 if positive, 128 if negative)
T1 = (T0−INT(T0))*256
M2 = INT(T1)
T2 = (T1−INT(T1))*256
M3 = INT(T2)
T3 = (T2−INT(T2))*256
M4 = INT(T3)

M1,M2,M3,M4 are the four mantissa byte values, M1 being the highest. Program 8 makes this conversion of a number input at the beginning of the program into the five bytes of a floating point format which are displayed on the screen. The program then checks by putting these values into the first variable in memory defined as a simple variable A in line 10.



```
5 REM ** REAL NUMBER FORMAT (PACKED) **
10 A=0
20 C=PEEK(45)+PEEK(46)*256+2
30 INPUTB
35 IFB=0THENPRINT0;0;0;0;0:PRINT:GOTO230
40 EX=INT(LOG(ABS(B))/LOG(2))
50 E=EX+129
60 R=B-2↑EX
70 SG=SGN(-B)*64+64
80 T0=(R/2↑EX)*128
90 M1=INT(T0)+SG
100 T1=(T0-INT(T0))*256
110 M2=INT(T1)
120 T2=(T1-INT(T1))*256
130 M3=INT(T2)
140 T3=(T2-INT(T2))*256
150 M4=INT(T3)
160 PRINTE;M1;M2;M3;M4
170 PRINT
180 POKEC,E
190 POKEC+1,M1
200 POKEC+2,M2
210 POKEC+3,M3
220 POKEC+4,M4
230 PRINTA
```

Program 8.

The following are examples of the storage of some floating point numbers:

Number	Exponent	M1	M2	M3	M4	Sign
1	$81	$80	$00	$00	$00	$00
−1	$81	$80	$00	$00	$00	$FF
.5	$80	$80	$00	$00	$00	$00
.25	$7F	$80	$00	$00	$00	$00
1E38	$FF	$96	$76	$99	$52	$00
1E−39	$00	$A0	$00	$00	$00	$00

The following are the principal routines within the interpreter which perform the arithmetic operations; all are usable by the programmer within machine code routines. These are all used by the Expression evaluation routine at $AD9E.

2.1.4 Evaluate expression

This is a long and very important routine which parses any expression, numeric or string, checking for syntax errors and evaluating the type of expression and result. The routine evaluates and expression whose starting address is pointed to by the charget pointers $7A,$7B. Since the routine involves a lot of stack processing, it first checks that there is sufficient space (it should be noted that long and complex expressions can generate an Out of memory error because of insufficient stack space). The expression type is determined and stored in location $0D. If $0D contains $FF then it is a string expression and if $0D contains $00 then it is a numeric expression. A series of routines then evaluates the expression and if it is numeric stores it in FAC#1. If it is a string expression then the string length is stored in the accumulator and the string pointer is in locations $64,$65. The result value or string is then assigned to the specified variable. If the variable is not found in the variable tables or arrays then it is created and the value or string allocated. The following are the entry points and functions of some of the routines used:

SADA9 – push .a to stack and run routine
$ADB8 – test for combination of <=> and store code in $4D
$ADD7 – process string operators
$AE20 – push argument in FAC#1 onto the stack. The stack format is:

1 . . $AD
2 . . $FA
3 . . operation address msb
4 . . operation address lsb
5 . . sign of value in FAC#1
6 . . value in FAC#1 lsb
7 . . value in FAC#1 2nd byte
8 . . value in FAC#1 3rd byte
9 . . value in FAC#1 msb
10 . . exponent in FAC#1
11 . . compare flag (from loc $4D)
12 . . operation hierarchy

The operation address is obtained from a table starting at $A080. This table also contains the operation hierarchy. This is stored in three bytes – hierarchy in one byte, and a two byte operation address. The operation hierarchy is derived from a hierarchy table at the start of the Basic interpreter. This places brackets and functions as the highest priority, followed by power, negate, */, +−, COMPARE, NOT, AND, OR. Bytes one and two of the stack are the return address and are fixed.

$AE58 – puts stack contents into FAC#2 and puts the exponent in .a
$AE83 – evaluation routine checks for ASCII numeric strings and operators
$AE83 – PI in floating point notation
SAEF1 – evaluates expression within brackets
$AEF7 – Syntax error if charget does not point to ')'
$AEFA – Syntax error if charget does not point to '('
$AEFD – Syntax error if charget does not point to ','
$AEFF – Syntax error if charget does not point to a byte identical to that in .a; if it does then .a returns with the next character

2.2 The arithmetic routines

The Basic interpreter includes twenty-four major arithmetic subroutines. These subroutines can be grouped into four categories; floating accumulator to memory transfers, floating accumulator to floating accumulator transfers, floating point to integer conversion and the actual arithmetic function routines. The following tables show the routines and how they can be used, parameters passed etc. It is recommended that anyone wishing to use these routines should first examine the full source code for these routines which is contained in Volume 1 of this series, *The Commodore 64 ROMs Revealed.*

Routine: Transfer FAC#1 to memory

Entry points:
$BBC7 – pack FAC#1 into $005C up
$BBCA – pack FAC#1 into $0057 up
$BBD0 – pack FAC#1 into current variable whose address is pointed to by locations $49,$4A
$BBD4 – pack FAC#1 into memory pointed to by .x and .y

Function: This routine compresses the six bytes of FAC#1 into five bytes by storing the sign byte as the most significant bit of the mantissa msb. These five bytes are then stored in a memory location pointed to by .x (lsb) and .y (msb) index registers.

Input parameters: No input parameters are required by entry points $BBC7, $BBCA or $BBD0.
.x index register – lsb of memory address pointer
.y index register – msb of memory address pointer

Output parameters:
Packed floating point value in memory, FAC#1 unchanged
Rounding flag in $70 set to zero

Registers used: Processor registers .a, .x, .y
FAC#1

Error messages: None

Example use: Example to transfer contents of FAC#1 to memory location starting at $C000.

```
LDX #$00  ;set .x to lsb address pointer
LDY #$C0  ;set .y to msb address pointer
JSR $BBD4 ;transfer
```

Routine: Transfer memory to FAC#1

Entry point: $BBA2

Function: This loads a value stored as a five byte floating point number, extracts a sign byte, and then stores it in the six bytes of FAC#1. The location of the value in memory is pointed to by the contents of .a (lsb) and .y (msb) registers.

Input parameters:
Accumulator – lsb of memory address pointer
.y index register – msb of memory address pointer

Output parameters:
FAC#1 contains the value which is still in memory
$70 (low order rounding byte) set to zero

Registers used:
Processor registers .a and .y
FAC#1

Error messages: None

Example use: This routine will load FAC#1 with the contents of memory starting at location $C000.

```
LDA #$00  ;lsb of address pointer
LDY #$C0  ;msb of address pointer
JSR $BBA2 ;transfer
```

Routine: Transfer memory to FAC#2

Entry point: $BA8C

Function: This takes the value stored as a five byte variable in memory at an address pointed to by .a (lsb) and .y (msb), unpacks the sign byte and stores the value in the six bytes of FAC#2.

Input parameters:
Accumulator – lsb of memory address pointer
.y index register – msb of memory address pointer

Output parameters: FAC#2 contains the value which is still stored in memory

Registers used:
Processor .a and .y registers
FAC#2

Error messages: None

Example use: Will take the floating point value in memory at location $C000 and transfer it to FAC#2.

```
     LDA #$00  ;lsb of address pointer
     LDY #$C0  ;msb of address pointer
     JSR $BA8C ;transfer
```

Note: To transfer FAC#2 to memory, FAC#2 must first be transferred to FAC#1 then FAC#1 transferred to memory.

Routine: Transfer FAC#1 to FAC#2

Entry point: $BC0F

Function: This moves the entire contents of FAC#1 into FAC#2, leaving both containing the same value.

Input parameters: FAC#1

Output parameters: FAC#2

Registers used: FAC#1 and FAC#2, registers .a and .x

Error messages: None

Example use: JSR $BC0F

Routine: Transfer FAC#2 to FAC#1

Entry point: $BBFC

Function: This moves the entire contents of FAC#2 into FAC#1, leaving both containing the same value.

Input parameters: FAC#1

Output parameters: FAC#2

Registers used: FAC#1 and FAC#2, processor registers .a and .x

Error messages: None

Example use: JSR $BBFC

Routine: Perform addition

Entry points:
$B867 - add FAC#1 to constant
$B86A - add FAC#1 to FAC#2

Function: The contents of FAC#1 are added to FAC#2 and the result stored in FAC#1. There are two entry points to this routine. The first at $B867 loads a five byte constant from memory pointed to by .a and .y into FAC#2 and adds it to FAC#1. The second at $B86A assumes that the two floating point numbers are already loaded into the two floating accumulators. The result is stored in FAC#1.

Input parameters: For entry point $B867
.a lsb memory address pointer to value B
.y msb memory address pointer to value B
FAC#1 contains value A
For entry point $B86A
FAC#1 contains value A
FAC#2 contains value B

Output parameters: FAC#1 contains the result of the addition

Registers used:
Processor registers .a, .x, .y
FAC#1 and FAC#2

Error messages: Overflow error if the sum of the two values exceeds the maximum or minimum size floating point value

Example use: To load two floating point values from memory and add them together leaving the result in FAC#1. The location of value A is $C000 and value B is $D000. FAC#1 is loaded using the routine at $BBA2.

```
LDA #$00  ;lsb of address of value A
LDY #$C0  ;msb of address of value A
JSR $BBA2 ;transfer value A from memory to FAC#1
LDA #$00  ;lsb of address of value B
LDY #$D0  msb of address of value B
JSR $B867 ;transfer value B to FAC#2 and perform addition
           and store the result in FAC#1
```

Routine: Perform subtraction

Entry points:
$B850 - subtract FAC#1 from constant
$B853 - subtract FAC#1 from FAC#2

Function: The contents of FAC#1 are subtracted from FAC#2 and the result stored in FAC#1. There are two entry points to this routine. The first at $B850 loads FAC#2 with a five byte value from memory pointed to by .a (lsb) and .y (msb). The other entry point at $B853 assumes that the two values are already loaded into the two floating accumulators. The result is stored in FAC#1.

Input parameters: For entry point $B850
.a lsb of address of value A
.y msb of address of value B

FAC#1 contains value B
For entry point $B853
FAC#1 contains value B
FAC#2 contains value A

Output parameters: FAC#1 contains the result

Registers used:
Processor registers .a, .y, .x
FAC#1 and FAC#2

Error messages: Overflow error if maximum or minimum floating point values are exceeded by the subtraction

Example use: To load two values stored in memory and subtract them leaving the result in FAC#1. Value A is stored at $C000 and is placed in FAC#1 by routine $BBA2. Value is stored at $D000. The result of subtracting value A from value B is stored in FAC#1.

```
LDA #$00  ;lsb of address of value A
LDY #$C0  ;msb of address of value A
JSR $BBA2 ;transfer A to FAC#1
LDA #$00  ;lsb of address of value B
LDY #$D0  ;msb of address of value B
JSR $B850 ;transfer B to FAC#2 and perform subtraction.
           Put the result in FAC#1.
```

Routine: Perform multiplication

Entry points:
$BA28 - multiply FAC#1 by constant
$BA2B - multiply FAC#1 by FAC#2

Function: The contents of FAC#1 are multiplied by the contents of FAC#2 and the result is stored in FAC#1. There are two entry points to this routine. The first at $BA28 loads a value into FAC#2 from memory pointed to by .a (lsb) and .y (msb) then multiplies FAC#1 by FAC#2. The second entry point at $BA2B assumes that both floating point accumulators have been loaded with the two values.

Input parameters: For entry point $BA28
.a lsb of address of value A
.y msb of address of value A
FAC#1 contains value B
For entry point $BA2B
FAC#1 contains value B
FAC#2 contains value A

Output parameters: FAC#1 contains the result

Registers used:
Processor registers .a, .x, .y
FAC#1 and FAC#2
Product area $26 to $2A

Error messages: Overflow error if the exponent of FAC#1 is $FF

Example use: This example loads two values stored in memory into the floating point accumulators, multiplies them together and puts the result in FAC#1. Value A is stored at $C000 and is placed in FAC#1 by routine $BBA2. Value B is stored at $D000. The result of multiplying A by B is stored in FAC#1.

```
LDA #$00  ;lsb of address of value A
LDY #$C0  ;msb of address of value A
JSR $BBA2 ;transfer A to FAC#1
LDA #$00  ;lsb of address of value B
LDY #$D0  ;msb of address of value B
JSR $BA30 ;transfer value B to FAC#2 and perform
           multiplication. Store the result in FAC#1.
```

Routine: Perform division

Entry points:
$BB0F - divide value in memory by FAC#1
$BB12 - divide FAC#2 by FAC#1

Function: This divides FAC#2 by FAC#1 and puts the result in FAC#1. The entry point $BB0F has the pointer to the five byte value stored in memory which must be transferred to FAC#2, the pointer is stored in .a (lsb) and .Y (msb), and .x must be loaded with the sign comparison byte - $6F. The contents of FAC#2 are then divided by the contents of FAC#1, loaded prior to the routine entry. The result is stored in FAC#1.

Input parameters: For entry point $BB0F
    .a lsb of memory address of value A
    .y msb of memory address of value A
    .x sign comparison byte from $6F
    FAC#1 contains value B
For entry point $BB12
    .a exponent of FAC#1 from $61
    FAC#1 contains value B
    FAC#2 contains value A

Output parameters: FAC#1 contains the result of dividing A by B

Registers used:
Processor registers .a, .x, .y
FAC#1 and FAC#2
Product area $26 to $2A

Error messages:
Division by zero error if FAC#1 = 0
Overflow error if FAC#1 exponent is $FF

Example use: This example loads two values from memory into the two floating point accumulators and divides the contents of FAC#2 by the contents of FAC#1 and stores the result in FAC#1. Value A is stored at $C000 and is placed in FAC#1 by the routine at $BBA2. Value B is stored at $D000.

```
LDA #$00  ;lsb of address of value A
LDY #$C0  ;msb of address of value A
JSR $BBA2 ;transfer A to FAC#1
LDA #$00  ;lsb of address of value B
LDY #$D0  ;msb of address of value B
JSR $BB0F ;transfer value to FAC#2 and divide B by A. Put
           result in FAC#1.
```

Routine: Calculate SIN

Entry point: $E26B

Function: The argument in radians is stored in FAC#1. It is evaluated and the sine of the angle stored in FAC#1.

Input parameters: FAC#1 contains the angle in radians

Output parameters: FAC#1 contains the sine of the angle

Registers used:
Processor registers .a, .x, .y
FAC#1

Error messages: None

Example use: Get the angle in radians from memory into FAC#1 using routine $BBA2, then convert it to a sine value.

```
LDA #$00  ;lsb of address of value
LDY #$C0  ;msb of address of value
JSR $BBA2 ;transfer value to FAC#1
JSR $E26B ;convert to sine and store in FAC#1
```

Routine: Calculate COS

Entry point: $E264

Function: The argument in radians stored in FAC#1 is converted to the cosine value which is stored in FAC#1. The routine actually adds PI/2 to the value and then calculates the sine.

Input parameters: FAC#1 contains the angle in radians

Output parameters: FAC#1 contains the cosine of the angle

Registers used:
Processor registers .a, .x, .y
FAC#1 and FAC#2

Error messages: None

Example use: Get the angle in radians from memory at $C000 into FAC#1 using the routine $BBA2, then convert it to cosine value.

```
LDA #$00  ;lsb of address of value
LDY #$C0  ;msb of address of value
JSR $BBA2 ;transfer value to FAC#1
JSR $E264  ;convert to cosine and store in FAC#1
```

Routine: Calculate TAN

Entry point: $E2B4

Function: This routine calculates the tangent of an angle in radians stored in FAC#1 and puts the result in FAC#1. The routine actually divides the sine of the value by the cosine of the value.

Input parameters: FAC#1 contains the angle in radians

Output parameters: FAC#1 contains the tangent of the angle

Registers used:
Processor registers .a, .x, .y
FAC#1 and FAC#2
Temporary floating accumulators at $4E and $57

Error messages: None

Example use: Get the angle in radians from memory at $C000 into FAC#1 using the routine $BBA2, then convert it to tangent value.

```
LDA #$00  ;lsb of address of value
LDY #$C0  ;msb of address of value
JSR $BBA2 ;transfer value to FAC#1
JSR $E2B4  ;convert to tangent and store in FAC#1
```

Routine: Calculate ATN

Entry point: $E30E

Function: The arc-tangent of a value stored in FAC#1 is calculated and the result in radians stored in FAC#1.

Input parameters: FAC#1 contains the value

Output parameters: FAC#1 contains the result in radians

Registers used:
Processor registers .a, .x, .y
FAC#1

Error messages: None

Example use: Get the value from memory at $C000 into FAC#1 using the routine at $BBA2, then convert it to radians and store it in FAC#1.

```
LDA #$00  ;lsb of address of value
```

```
LDY #$C0  ;msb of address of value
JSR $BBA2 ;transfer value to FAC#1
JSR $E30E ;convert to radians and store in FAC#1
```

Routine: Calculate EXP

Entry point: $BFED

Function: This routine calculates the exponent (the value of E to the power of the value in FAC#1) and stores the result in FAC#1.

Input parameters: FAC#1 contains the value

Output parameters: FAC#1 contains the exponent of the value

Registers used:
Processor registers .a, .x, .y
FAC#1 and FAC#2

Error messages: Overflow error if the value of the exponent is greater than 88.029

Example use: Get the value from memory at $C000 into FAC#1 using routine $BBA2, then calculate the exponent.

```
LDA #$00  ;lsb of address of value
LDY #$C0  ;msb of address of value
JSR $BBA2 ;transfer value to FAC#1
JSR $BFED ;calculate exp and put in FAC#1
```

Routine: Calculate LOG

Entry point: $B9EA

Function: This performs the calculation of the log to the base E of a value in FAC#1 and stores the result in FAC#1.

Input parameters: FAC#1 contains the value

Output parameters: FAC#1 contains the log of the value

Registers used:
Processor registers .a, .x, .y
FAC#1 and FAC#2
Product area $26 to $2A

Error messages: Illegal quantity if value is zero or minus

Example use: Get the value from memory at $C000 into FAC#1 using routine $BBA2, then calculate the log of the value and put the result in FAC#1.

```
LDA #$00  ;lsb of address of value
LDY #$C0  ;msb of address of value
JSR $BBA2 ;transfer value from memory to FAC#1
JSR $B9EA ;calculate log and put result in FAC#1
```

Routine: Calculate power

Entry points:
$BF78 - raise FAC#2 to power of constant in memory
$BF7B - raise FAC#2 to power of FAC#1

Function: The contents of FAC#2 are raised to the power of the value stored in FAC#1. Before using this routine FAC#2 must be loaded. If either value is zero then FAC#1 is loaded with either 0 or 1 depending on which FAC was zero. The evaluation is performed by saving FAC#1 to zero page and then multiplying the logarithm of FAC#2 by FAC#1 and getting the exponent of the result. There are two entry points. The first at $BF78 raises FAC#2 to the power of a constant stored in memory and pointed to by .a (lsb) and .y (msb). The second entry point requires the values to be in FAC#1 and FAC#2.

Input parameters: For entry point $BF78
  .a lsb of power value in memory
  .y msb of power value in memory
  FAC#2 - value to be raised to the power of constant
For entry point $BF7B
  FAC#1 - value of power
  FAC#2 - value to be raised to the power of FAC#1

Output parameters: FAC#1 contains the result

Registers used:
Processor registers .a, .x, .y
FAC#1 and FAC#2
Product register $26 to $2A
Miscellaneous work area $4e to $53

Error messages: No error message is given if one of the FACs contains a zero. This error is flagged by the contents of FAC#1, which contains zero if the power is zero and one if the value is zero. (*Note:* This is a potential source of error in a program.)
Illegal quantity error if either number is negative and the value is not an integer. If the result is too large an Overflow error is generated.

Example use: Get a value from memory at $C000 into FAC#2 using the routine at $BA8C, then raise it to the power of a value stored at $D000, and put the result in FAC#1.

```
LDA #$00  ;lsb of address of value A
LDY #$C0  ;msb of address of value A
JSR $BA8C ;transfer to FAC#2
LDA #$00  ;lsb of address of power value B
LDY #$D0  ;msb of address of power value B
JSR $BF78 ;raise value B to the power of A and put the result in
           FAC#1
```

Routine: Calculate SQR

Entry point: $BF71

Function: The contents of FAC#1 (the argument) are transferred to FAC#2. FAC#1 is then loaded with .5 and the routine jumps to the perform power routine at $BF78. The result is stored in FAC#1.

Input parameters: FAC#1 contains the argument.

Output parameters: FAC#1 contains the result

Registers used:
Processor registers .a, .x, .y
FAC#1 and FAC#2
Product register $26 to $2a
Miscellaneous work area $4e to $53

Error messages: Illegal quantity error if it is a minus value

Example use: Get a value from memory at $C000 into FAC#1 using routine $BBA2, then find its square root and put the result in FAC#1.

```
LDA #$00  ;lsb of address of value of argument
LDY #$C0  ;msb of address of value of argument
JSR $BBA2 ;transfer value to FAC#1
JSR $BF71 ;calculate sqr of value and put result in FAC#1
```

Routine: Fixed point to floating point number conversion

Entry point: $B391

Function: This routine converts a two byte integer held in .a (msb) and .y (lsb) into its floating point equivalent. This value is stored in FAC#1.

Input parameters:
.a msb of integer value
.y lsb of integer value

Output parameters:
FAC#1 contains the floating point equivalent
Variable type flag in $0D is set to 0

Registers used:
Processor registers .a, .x, .y
FAC#1

Error messages: None

Example use: Convert the 16 bit integer value $B7FE to floating point value in FAC#1.

```
LDA #$B7  ;msb of integer value
LDY #$FE  ;lsb of integer value
JSR $B391 ;convert to floating point in FAC#1
```

Routine: Floating point to fixed point number conversion

Entry point: $BC9B

Function: The floating point number is stored in FAC#1 and is converted to a two byte integer value which is stored in locations $65 (lsb) and $66 (msb). If the value in FAC#1 is greater than +32767 or less than −32768 then the overflow is stored in $68.

Input parameters; FAC#1 contains the floating point value

Output parameters:
$65 - lsb of integer value
$66 - msb of integer value
$68 - overflow if value exceeds maximum integer value

Registers used:
Processor registers .a, .y, .x
FAC#1

Error messages: None

Example use: Convert a five byte floating point value in memory at address $C000 to a two byte integer in .a (msb) and .y (lsb). The value is first moved to FAC#1, then converted to an integer value in $65,$66. These are transferred to .a and .y.

```
LDA #$00  ;lsb of address of floating point value
LDY #$C0  ;msb of address of floating point value
JSR $BBA2 ;transfer value to FAC#1
JSR $BC9B ;convert to integer
LDA $66   ;put integer msb in .a
LDY $65   ;put integer lsb in .y
```

Routine: Convert the value stored as a string to floating point value

Entry point: $BCF3

Function: The value stored as a string which is to be converted is stored in memory at a location pointed to by the charget program pointers $7A and $7B. The numeric value stored in the string is checked then converted to floating point form in FAC#1.

Input parameters:
$7A - lsb of address of start of string
$7B - msb of address of start of string
The string is located in memory starting at an address pointed to by the above two parameters. The string is unchanged by this routine.

Output parameters: FAC#1 contains the floating point equivalent of the string

Registers used:
Processor registers .a, .x, .y
FAC#1

Error messages: Overflow error if the value in FAC#1 is too large or small

Example use: Convert a value stored as a string at starting address $C000 into a floating point value in FAC#1.

```
LDA #$00  ;lsb of address of string start
STA $7A   ;store in charget pointer lsb
LDA #$C0  ;msb of address of string start
STA $7B   ;store in charget pointer msb
JSR $BCF3 ;convert string to floating point in FAC#1
```

Routine: Convert a floating point number into a string

Entry point: $BDDD

Function: The value stored in FAC#1 is converted into an ASCII string stored in a buffer starting at location $0100. On exit from the routine a zero terminating byte is placed at the end of the string and the buffer start address is stored in .a (lsb) and .y (msb). This is required to set the correct input parameters for the print string routine at $AB1E.

Input parameters: FAC#1 contains the floating point value

Output parameters: Buffer starting at $0100 contains the string
.a – lsb of buffer start address
.y – msb of buffer start address

Registers used:
Processor registers .a, .x, .y
FAC#1

Error messages: None

Example use: Get the floating point value from memory at $C000 into FAC#1 and convert it to a string stored in the buffer at $0100. This string is then displayed on the screen using routine $AB1E.

```
LDA #$00   ;lsb of address of value
LDY #$C0   ;msb of address of value
JSR $BBA2  ;transfer floating point value to FAC#1
JSR $BDDD ;convert to a string in $0100 up
JSR $AB1E  ;display string on current output device
```

Routine: Compare the contents of FAC#1 with a value in memory

Entry point: $BC5B

Function: The value stored in FAC#1 is compared with a five byte floating point value stored in memory at a location pointed to by .a (lsb) and .y (msb). On exit the accumulator contains the comparison flag: $00 = that both values are the same; $01 = that FAC#1 is greater than the value in memory; and $FF = that FAC#1 is less than the value in memory.

Input parameters: FAC#1 contains the floating point value A
.a lsb of address of floating point value in memory
.y msb of floating point value in memory

Output parameters: .a contains the comparison flag

Registers used:
Processor registers .a, .x, .y
FAC#1

Error messages: None

Example use: Get a floating point value into FAC#1 from memory at $C000 and compare it with a floating point value in memory at $D000. Store the comparison flag in location $12.

```
LDA #$00  ;lsb of address of value A
LDY #$C0  ;msb of address of value A
JSR $BBA2 ;transfer value A to FAC#1
LDA #$00  ;lsb of address of value B
LDY #$D0  ;msb of address of value B
JSR $BC5B ;compare value A to value B
STA $12   ;save comparison flag in location $12
```

Routine: Complement the contents of FAC#1

Entry point: $B947

Function: This routine replaces the contents of FAC#1 by its twos complement. This means that all the zeros are converted to ones and vice versa, then one is added to the result.

Input parameters: FAC#1 contains the value to be complemented

Output parameters: FAC#1 contains complemented value

Registers used:
Processor registers .a, .x, .y
FAC#1

Error messages: None

Example use: Get the value into FAC#1 from memory at $C000 and complement it. The result is stored in FAC#1.

```
LDA #$00  ;lsb of address of value
LDY #$C0  ;msb of address of value
JSR $BBA2 ;transfer value to FAC#1
JSR $B947 ;complement FAC#1 and store result in FAC#1
```

Routine: Round FAC#1

Entry point: $BC1B

Function: The exponent of FAC#1 in byte $61 is tested. If the content is zero then the routine exits; if not then the rounding byte in $70 is multiplied by two and the state of the carry flag checked. If carry is clear then it exits. Otherwise the floating point value is incremented by 1.

Input parameters: FAC#1 contains the value

Output parameters: FAC#1 contains the rounded floating point value

Registers used:
Processor registers .a
FAC#1

Error messages: Overflow error if rounding makes the value too large or small

Example use: Get the floating point value into FAC#1 from memory at $C000, then round it and leave the rounded value in FAC#1.

```
LDA #$00  ;lsb of address of value
LDY #$C0  ;msb of address of value
JSR $BBA2 ;transfer value to FAC#1
JSR $BC1B ;round value in FAC#1
```

2.3 Using the arithmetic routines in a machine code program

Using the arithmetic routines within the Basic interpreter can save the programmer a lot of time in program development. It can also greatly reduce the size of a machine code program. The only penalty is that in any program using eight or sixteen bit values the interpreter routines will have a considerably slower run time than specially written routines. When faced with the necessity of having to use arithmetic routines the best procedure is always to use the interpreter routines and replace these only if the program is running too slowly.

The best way of learning to use these routines, in addition to actually trying to use them, is to study some of the routines in this book which utilise them, in particular the matrix calculation routines in Chapter 5. It is also an excellent idea to examine the annoted assembly listings of any routine you intend using; these annotated listings are contained in *The Commodore 64 ROMs Revealed* in this series.

It is quite simple to utilise the interpreter arithmetic routines within a machine code program. The essential point to remember is that the interpreter does all its calculations on floating point numbers, therefore all integer values must first be converted to floating point. The following is an example of a routine using the interpreter arithmetic routines:

calculation #C = (A+22) / (B*5)

Where values A and B are both positive unsigned sixteen bit integer values, these are both input from the keyboard at the beginning of the routine and the result C is a five byte floating point value which is both stored in memory and displayed on the screen.

Variable storage locations in memory used by this routine are:

$C000 – lsb of value A
$C001 – msb of value A
$C002 – lsb of value B
$C003 – msb of value B
$C004 to $C008 – temporary floating point value storage 1
$C009 to $C00D – temporary floating point value storage 2
$C00E to $C012 – floating point result C storage

```
033C        !CALCULATE (A+22)/(B*5)
033C        ! WHERE A AND B ARE INPUT FROM
033C        ! THE KEYBOARD.
033C        ! ENTRY AT SYS 49171.
033C        !
033C        ! RESULT IS PRINTED
033C        !
C000        *=$C000
C000 0000   AV          WOR 0
C002 0000   BV          WOR 0
C004 000000 TF1         BYT 0,0,0,0,0
C009 000000 TF2         BYT 0,0,0,0,0
C00E 000000 TF3         BYT 0,0,0,0,0
C013 A000   ENTRY       LDY #$00
C015 20CFFF L1          JSR $FFCF                  !INPUT BYTE
C018 C90D               CMP #$0D                   !CARRIAGE RETURN?
C01A F006               BEQ L2                     !YES
C01C 990002             STA $0200,Y                !STORE BYTE
C01F C8                 INY                        !DO NEXT
C020 D0F3               BNE L1                     !ALWAYS
C022 A900   L2          LDA #$00                   !ZERO TERMINATOR
C024 990002             STA $0200,Y
C027 A900               LDA #$00                   !SET CHARGET TO
C029 857A               STA $7A                    !BUFFER
C02B A902               LDA #$02
C02D 857B               STA $7B
C02F 207900             JSR $0079
C032 208AAD             JSR $AD8A                  !CONVERT TO # 0-65535
C035 20F7B7             JSR $B7F7                  !MAKE INTEGER
C038 A514               LDA $14                    !STORE VALUE
C03A 8D00C0             STA AV                     !IN TEMP
C03D A515               LDA $15
C03F 8D01C0             STA AV+1
C042 A000   ENTRY1      LDY #$00
C044 20CFFF L3          JSR $FFCF                  !INPUT BYTE
C047 C90D               CMP #$0D                   !CARRIAGE RETURN?
C049 F006               BEQ L4                     !YES
C04B 990002             STA $0200,Y                !STORE BYTE
C04E C8                 INY                        !DO NEXT
C04F D0F3               BNE L3                     !ALWAYS
C051 A900   L4          LDA #$00                   !ZERO TERMINATOR
C053 990002             STA $0200,Y
C056 A900               LDA #$00                   !SET CHARGET TO
C058 857A               STA $7A                    !BUFFER
C05A A902               LDA #$02
C05C 857B               STA $7B
C05E 207900             JSR $0079
C061 208AAD             JSR $AD8A                  !CONVERT TO # 0-65535
C064 20F7B7             JSR $B7F7                  !MAKE INTEGER
C067 A514               LDA $14                    !STORE VALUE
C069 8D02C0             STA BV                     !IN TEMP
```

```
C06C A515          LDA $15
C06E 8D03C0        STA BV+1
C071 AD01C0        LDA AV+1         !GET FIRST VALUE
C074 AC00C0        LDY AV
C077 2091B3        JSR $B391        !FLOAT IT
C07A A204          LDX #<TF1        !STORE IN TEMP FAC1
C07C A0C0          LDY #>TF1
C07E 20D4BB        JSR $BBD4
C081 A900          LDA #$00         !VALUE 22 ($16)
C083 A016          LDY #$16
C085 2091B3        JSR $B391        !FLOAT IT
C088 A904          LDA #<TF1        !POINT TO TEMP
C08A A0C0          LDY #>TF1        !FAC1
C08C 2067B8        JSR $B867        !ADD
C08F A204          LDX #<TF1        !STORE IN TEMP FAC1
C091 A0C0          LDY #>TF1
C093 20D4BB        JSR $BBD4
C096 AD03C0        LDA BV+1         !GET SECOND VALUE
C099 AC02C0        LDY BV
C09C 2091B3        JSR $B391        !FLOAT IT
C09F A209          LDX #<TF2        !STORE IN TEMP FAC2
C0A1 A0C0          LDY #>TF2
C0A3 20D4BB        JSR $BBD4
C0A6 A900          LDA #$00         !GET VALUE 5
C0A8 A005          LDY #$05
C0AA 2091B3        JSR $B391        !FLOAT IT
C0AD A909          LDA #<TF2        !POINT TO TEMP
C0AF A0C0          LDY #>TF2        !FAC2
C0B1 2028BA        JSR $BA28        !MULTIPLY
C0B4 A904          LDA #<TF1        !POINT TO TEMP
C0B6 A0C0          LDY #>TF1        !FAC1
C0B8 200FBB        JSR $BB0F        !DIVIDE
C0BB A20E          LDX #<TF3        !STORE RESULT IN
C0BD A0C0          LDY #>TF3        !TEMP FAC3
C0BF 20D4BB        JSR $BBD4
C0C2 20DDBD        JSR $BDDD        !CONVERT TO STRING
C0C5 201EAB        JSR $AB1E        !PRINT STRING
C0C8 4C74A4        JMP $A474        !'READY.'
```

Program 9.

Chapter Three

The Keywords of BASIC

ABS

Abbreviated entry: A(shift)B

Token: Hex $B6 Decimal 182

Modes: Direct and program

Purpose: The arithmetic expression contained in brackets following the ABS command is converted to its absolute value. This means that the value is always returned as a positive value.

Syntax: ABS (arithmetic expression). ABS can appear within a logical expression, in a PRINT statement and to the right of an assignment statement.

Errors: This routine can generate a number of errors; these are the result of either an invalid arithmetic expression or a non arithmetic expression.

Syntax error – wrong command syntax, e.g. missing closing bracket
Overflow error – result of expression evaluation which is too large
Division by zero – attempt within the expression to divide by zero
Type mismatch – using a non arithmetic expression

Use: This command has fairly limited applications, all confined to numerical operations.

ROM routine entry point: $BC58

Routine operation: The routine is very short (three bytes !) and simply takes the sign byte of FAC#1, in location $66, and on it performs a logical shift right, thereby ensuring that it always contains a positive flag.

AND

Abbreviated entry: A(shift)N

Token: Hex $AF Decimal 175

Modes: Direct and program

Purpose: This command performs a logical AND between two expressions. These expressions are first converted into double byte integer values, an AND performed, and the result returned as a two byte integer.

Syntax: Expression A AND expression B. The expression can be either arithmetic or logical but must always be either an integer value or a floating point value within the range +32767 and −32768.

Errors: There are several errors associated with this command:
Syntax error - incorrect command syntax
Illegal quantity - if expressions exceed maximum/minimum values
Type mismatch - using a non arithmetic or logical expression

Use: The AND command acts either as a logical operator or as a bitwise operator on two straight 16 bit values.

As a logical operator the AND command is used to ensure that two conditions are met before a particular operation is performed, as in the following example:

```
IF A =22 AND B=5 THEN PRINT "TEST O.K."
```

The result of a comparison gives −1 if the comparison is true and 0 if it is false. If a comparison is true then a value of −1 is returned by the comparison routine. This is represented as a twos complement value with a binary representation of:

1111 1111 1111 1111 or hex $FFFF or −1

Similarly a false comparison returns a value of zero, represented as:

0000 0000 0000 0000 or hex $0000or 0

Therefore an AND will give a true condition only when both conditions are true (both values are $FFFF); all other states will be regarded as false.

A bitwise AND compares the first bit of one value with the first bit of the second value and gives a result according to the following truth table:

AND

	1	0
1	1	0
0	0	0

Thus the command:

1278 AND 3279

has as its binary equivalent:

```
    0000 0100 1111 1110
AND 0000 1100 1100 1111
```

This gives the result:

0000 0100 1100 1110

or decimal 1230.

It should be noted, of course, that the AND operation is performed on two signed double byte integers. These are stored in twos complement form. Thus a value of −1 has a binary equivalent of 1111 1111 1111 1111 and any number ANDed with −1 will always return the same number. Likewise a positive value ANDed with a negative will always give a positive result.

The hierarchy of logical operators is NOT, AND, OR, thus NOT always has a higher priority than AND.

ROM routine entry point: $AFE9

Routine operation: The two arguments in floating point format are stored in FACH#1 and FAC#2. They are first converted to fixed point integer values, the AND operation performed on the two 16 bit numbers, and the result converted back from integer to floating point form in FAC#1.

ASC

Abbreviated entry: A(shift)S

Token: Hex $C6   Decimal 198

Modes: Direct and program

Purpose: This command returns the ASCII code value of the first character in a string expression.

Syntax: ASC (string expression). The string expression can be any valid string expression either variable, literal or function including string concatenation. The exception is a null string which will return an Illegal quantity error, the reason being that such a null string (this is represented by "") has a length of zero.

Errors: Syntax error – wrong command syntax e.g. missing closing bracket
Type mismatch – use of a non string expression
Illegal quantity error – null string expression

Use: This command is useful in any situation where it is required to convert a character into its corresponding value. It is particularly useful for trapping or validating individual characters within strings such cursor control, insert/delete and carriage return characters.

ROM routine entry point: $B78B

Routine operation: The routine first gets the string length and any string with a zero length is rejected with an Illegal quantity error. The .y index register is loaded with the character which is pointed to by locations $22,$23. This character is then converted to a floating point number stored in FAC#1.

ATN

Abbreviated entry: A(shift)T

Token: Hex $C1 Decimal 193

Modes: Direct and program

Purpose: This calculates the angle where the tangent of that angle is known. The angle is returned in radians.

Syntax: ATN (arithmetic expression). Any arithmetic expression can be used.

Errors: Syntax error – wrong command syntax e.g. missing closing bracket
Type mismatch – non arithmetic expression
Overflow error – if expression is outside floating point range

Use: This command is useful in many trigonometric applications. It should, of course, be noted that the returned value is in radians and not degrees; to convert to degrees multiply by 180/pi.

ROM routine entry point: $E30E

Routine operation: The tangent is stored in FAC#1 from where it is converted to the equivalent angle in radians which is also stored in FAC#1.

CHR$

Abbreviated entry: C(shift)H

Token: Hex $C7 Decimal 199

Modes: Direct and program

Purpose: This command generates a character from its equivalent ASCII code number.

Syntax: CHR$ (numeric expression). The expression within the brackets must, when evaluated, be within the range 0 to 255.

Errors: Syntax error – wrong command syntax e.g. missing closing bracket
Type mismatch – non numeric expression
Illegal quantity – expression is outside range 0 to 255

Use: This command is the reverse of the ASC command and has similar applications. This command is particularly useful when adding editor or colour control characters to strings.

CHR$ can be used to convert values stored in memory (and accessed using PEEK) into string characters for display on the screen, or for use within the program. Since the CBM 64 does not use a standard ASCII character set another application is to use ASC to convert each character to its CBM ASCII code value, perform the required code conversion, and then use CHR$ to convert back to the corresponding string character. This application is essential when using some non CBM printers or when communicating with other makes of computer. It could also be used in encoding and enscription routines.

The command CHR$ has one oddity; the use of CHR$(0) allows the addition of a null character with length 1 to a string. The null character will never be printed but will register when LEN is used.

ROM routine entry point: $B6EC

Routine operation: The single byte parameter is input and evaluated and checked for correct range (0–255) by the routine at $B7A1. A single character string space is then allocated, the character generated from the input parameter is stored in this string space and the string pointers are set up in the allocated string variable.

CLOSE

Abbreviated entry: CL(shift)O

Token: Hex $A0 Decimal 160

Modes: Direct and program

Purpose: This command is used to inform the computer that the processing of a file is completed. The processor then deletes reference to the file from its file tables, depending on which output device is being accessed. The CLOSE command also sets various end of file pointers.

Syntax: CLOSE file number. The file number must be a value between 1 and 255.

Errors: Syntax error – if there is no file number
Illegal quantity – if the file number is outside the range 1–255
Note: No error is generated if the file does not exist

Use: The CLOSE command deletes the file entries from the file tables set up by the OPEN command (see OPEN command for details of file tables). If the files opened were to either the screen (device 3) or keyboard (device 0) then no other action is taken. When closing cassette files which have been used to write data the last buffer is dumped to tape and an end of tape header is written containing the end of tape value 5. Serial files, previously opened for write, when closed send the buffer contents to the serial device media together with an end of file command. This causes the serial device to close the file and set/reset any pointers within the serial device (see *The Commodore 64 Disk Drive Revealed* for details on the functioning of the disk commands and the disk internal operating system). Serial and cassette devices opened for read will simply clear the input buffer.

ROM routine entry point: vector indirect entry $031C
routine entry $F291

Routine operation: The logical file number of the file to be closed is passed in the processor accumulator. Keyboard, screen and unopened files just pass straight through the routine, but tape files open for write are closed by dumping the last buffer and conditionally writing an end of tape block. Serial files are closed by sending a close file command if a secondary address was specified in the open command.

CLR

Abbreviated entry: C(shift)L

Token: Hex $9C Decimal 156

Modes: Direct and program

Purpose: Resets the variable pointers so that all variables are in practice erased while leaving the Basic program unchanged.

Syntax: CLR. This command has no parameters.

Errors: Will produce errors only if the programmer has been changing the variable pointers.

Use: This command does not in fact erase any of the variables by replacing them with nulls; instead it simply restores the variable pointers. Thus the start of an arrays pointer will contain the top of the Basic program address plus two, and the bottom of the strings pointer is set to the top of the memory pointer. The pointer to DATA statements is also cleared. The temporary string stack is cleared and the main stack is also cleared. The fact that the CLR command resets the processor stack pointer to the bottom of the stack means that although CLR can be used in program mode it will remove all loop returns. Therefore if CLR is performed within a GOSUB or FOR ... NEXT loop, then the program will fail on the RETURN or NEXT command. In many applications it is preferable to use POKE commands to change just the required pointers rather than use the CLR command. It should also be noted that the CLR command will do a partial close on all files to cassette or serial which are open; this will result in loss of data and the erasing of all open files from the file tables.

ROM entry point: $A65E

Routine operation: The CLR function first checks that there is no following parameter, then sets string pointers $33,$34 equal to the top of memory pointers $37,$38 and the start of arrays $2F,$30 equal to the start of variables $2D,$2E thus erasing all variable storage pointers. The I/O pointers are returned to default values and the stack pointer reset to remove unwanted stack variables. The routine performs a restore and blocks the CONT command.

CMD

Abbreviated entry: C(shift)M

Token: Hex $9D   Decimal 157

Modes: Direct and program

Purpose: This command is used to set the primary output device to a previously opened file rather than the screen. All output following the CMD command will then be directed to the new output device.

Syntax: CMD logical file number, string. The file number must be a value between 1 and 255. The comma separator between the file number and the string is only necessary if a string is included within the CMD command.

Errors: Syntax error – wrong command syntax e.g. no file number
Illegal quantity – logical file number exceeds the limits of 1 to 255
File not open – if the specified logical file is not opened

Use: When this command has been used all PRINT or LIST commands will send data to the device specified in the previous OPEN command. This will continue until a PRINT# file number command resets the output to the screen. It then uses a CLOSE command to close the file. An alternative method in direct mode is to perform any operation which will generate a syntax error; this will reset output to the default device. This should then be followed by a blank line output to 'unlisten' the output device.

ROM routine entry point: $AA86

Routine operation: The parameter following the CMD command is evaluated by the routine at $B79E which gets a single byte parameter. The result is stored in the .x index register, the output device number variable in location $13 is then set to the value in x and PRINT is performed.

CONT

Abbreviated entry: C(shift)O

Token: Hex $9A Decimal 154

Mode: Direct only – attempting to use CONT within a program will result in an endless loop within CONT and therefore a program crash.

Purpose: To restart the execution of a Basic program after either pressing the STOP key or the program encountering a STOP command.

Errors: Can't continue error – on using CONT after an execution error or after changing the program or using CLR

Use: The main use of CONT is in debugging a Basic program. By inserting STOP commands at strategic points within the program one can stop the program, examine all the variables in direct mode and then resume operation with CONT. While the program is stopped its variables can also be changed in the direct mode; however new variables or lines cannot be added.

ROM routine entry point: $A857

Routine operation: This routine restores the line address pointer in chargot at locations $7A,$7B using the contents of the pointer to the Basic statement for the CONT variable at $3D,$3E. It also sets the current line number variable in $39,$3A equal to the previous line number in $3B,$3C. If, however, the contents of $3E are zero then a Can't continue error is generated.

COS

Abbreviated entry: None

*Token:*Hex $BE Decimal 190

Mode: Direct and program

Purpose: This command evaluates the cosine of an angle in radians.

Syntax: COS (arithmetic expression). The expression must be syntactically correct and within the range permissible for floating point numbers.

Errors: Syntax error – wrong command syntax e.g. missing closing bracket
Type mismatch – non arithmetic expression
Overflow error – expression is outside the permissible floating point

Use: This command is used within many trigonometric applications. It should be noted that the value of the expression must be in radians rather than degrees. An angle can be converted to radians by multiplying the angle by pi/180.

ROM routine entry point: $E264

Routine operation: The argument in radians is stored in FAC#1, this is then added to a value of pi/2 stored in FAC#2 and the result stored in FAC#1. The routine then jumps to the perform SIN routine at $E26B and the result is stored in FAC#1.

DATA

Abbreviated entry: D(shift)A

Token: Hex $83 Decimal 131

Mode: Program mode only

Purpose: This command allows data to be stored within a program without the necessity of it being entered separately from the keyboard, tape or disk. The data, which can be any alphanumeric or ASCII character values or strings, is then accessed using the READ command.

Syntax: DATA followed by ASCII characters. Two delimiters are used in a DATA statement. A " is used to delimit string data, and a comma is used to separate each item of data. A colon encountered on the same line as a DATA statement signifies the end of the data.

Errors: None

Use: DATA statements are a very useful way of storing data, in particular constants, within a program. Though DATA statements can be placed anywhere within the program the order of data within these statements is important, since the READ command sequentially accesses the data. The data pointer can be reset to the beginning of all DATA statements only by the RESTORE command. To access DATA statements in a random manner one would need to know the start address of each data statement as it is stored within the program memory, and use these addresses to put into the pointers to the current DATA statement variable in locations $41,$42. This would then cause a READ to get the desired data. Data statements can be forced into a program using the keyboard buffer to emulate the entry of a program line (see Chapter 2 for details of this and Chapter 4 for a Restore to line # routine).

ROM routine entry point: $A8F8

Routine operation: This routine is part of the RETURN routine and is used to search for the next Basic statement following the DATA statement, thereby ignoring the data following the DATA statement. The main associated data accessing routine is the READ routine.

DEF FN

Abbreviated entry: DEF is D(shift)E
FN has no abbreviation

Token: DEF Hex $96 Decimal 150
FN Hex $A5 Decimal 165

Modes: Program mode only

Purpose: This command is used to assign a user defined function which can be called later within the program by FN. The function can consist of any valid mathematical formula.

Syntax: DEF FN floating point variable (floating point variable) = arithmetic expression. The function definition must precede the FN call within a program and must fit within a single Basic line.

Errors: Syntax error – wrong command syntax e.g. non floating point variable (It should be noted that this error will be produced on the line using the FN rather than the DEF FN line)

Type mismatch – use of string variables
Division by zero – attempt to divide by zero within an expression
Out of memory – recursive calling of function by function
Overflow error – result of an expression evaluation which is too large or small
Undefined function – FN call before DEF FN definition

Use: The principal use of the DEF FN command is to save program space and complexity by allowing a complex formula, used several times within a program, to be defined just once. In fact DEF FN acts rather like a special subroutine jump. It could be replaced by a jump to a subroutine but this would be considerably slower, and would only be justifiable if the expression required more than a single line of Basic program to define it. The function defintion is stored as a simple variable (see Chapter 2 for details on how this variable is stored). It should be noted that the variable in brackets does not change when a function is called. Although it is used by the function definition it is temporarily stored in an area of memory reserved for the function definition. Since the function definition is stored as a variable it can be redefined at any time within a program; similarly one function definition can call another function as its variable.

ROM routine entry points: perform DEF is at $B3B3
perform FN is at $B3F4
check FN syntax at $B3E1

Routine operation: DEF – a syntax check is first carried out using the routine at $B3E1. The mode of operation is then checked to make sure that it is in program mode, and a left bracket is searched for. If found, then the following variable is located in memory using routine $B08B. A right bracket is then checked for and the next character in Basic tested to make sure that it is an = sign. The five bytes of data obtained are then pushed onto the stack in the following format:

(1) function token of the first character in the variable name

(2) variable address pointer from locations $47
(3) and $48
(4) pointer to Basic for charget from $7A
(5) and $7B

The evaluate FN function first calls the routine at $B3E1 which checks syntax and then gets the variable address. The expression is evaluated and the result stored in FAC#1. The data placed on the stack by the DEF routine is recovered and stored in RAM memory at a location pointed to by the values in locations $4E,$4F.

Both routines call a routine to check the FN syntax. This first checks for the FN token, $A5, then sets the function flag in location $10 with the OR of the function name AND $80. If the function exists it is searched for; if not then it is set up. Finally the routine checks that the value is numeric.

DIM

Abbreviated entry: D(shift)I

Token: Hex $86 Decimal 134

Mode: Direct and program

Purpose: This command allocates space in memory for the storage of an array of specified name, number of dimensions, number of elements in each dimension and variable type.

Syntax: DIM name (arithmetic expression 1, arithmetic expression 2, ..., arithmetic expression n) [,name 2 (arithmetic expression 1, ...)]. The square brackets indicate optional repetitions. Each expression is evaluated and converted to a two byte positive integer which must be within the range 0 to 32767 (though high values like 32767 will always give an Out of memory error). It is usually best to dimension arrays at the beginning of a program, and any attempt to put a DIM statement within a loop or create a new array using the same variables will always give a Redimensioned array error.

Errors: Syntax error – wrong command syntax
Out of memory – number of elements is too large for the available memory
Redim'd array – attempting to redefine an existing array
Illegal quantity – number of elements is less than 0 or greater than 32767

Use: This is a very straightforward command which must be used before setting up an array. It is possible to use subscripted variables without having defined the array with a DIM, in which case the number of elements in each dimension will default to 11. (*Note:* DIM A (10) gives an array with eleven elements since the zero element is used.) In a default array any attempt to use more than three dimensions will give an Out of memory error due to the fact that the default array will be 10,10,10,10 and this uses more memory than is available on the CBM 64. For further details on how arrays are stored see Chapter 2.

ROM routine entry point: $B081

Routine operation: The presence of a variable of the same name is first checked using the routine at $B090. If one is not found then the routine sets up an array with the variable name and number of elements specified in the DIM statement. It checks to see if charget points to a comma as the next character; if so then the routine loops back and repeats the procedure for the next specified array.

END

Abbreviated entry: E(shift)N

Token: Hex $80 Decimal 128

Modes: Direct and program

Purpose: Informs the computer that it has reached the end of the program, whereupon it exits to the direct mode and prints a Ready message. The CONT command can be used to resume execution after an END statement.

Syntax: END has no parameters but must always be followed by a colon or end of line marker.

Errors: Syntax error - following END by a parameter

Use: This command is used to halt execution of the program, a function it shares with the STOP command. The END command is not essential at the end of a program if the end is at the highest program line number, but is essential if the program is to end prior to that. Like the STOP command, END can be used to set break points in a program during debugging, where a CONT will resume program execution.

ROM routine entry point: $A82C

Routine operation: This routine is called by either the STOP key detect routine at $FFE1, the routine at $A7BE which detects the terminating double zero bytes of a Basic program, or by the keyword END. Which action is performed depends on the state of the Z and carry flags in the processor status register. If carry and Z are both set then a STOP break is initiated; if carry is clear then END is performed.

EXP

Abbreviated entry: E(shift)X

Token: Hex $BD Decimal 189

Modes: Direct and program

Purpose: Calculates e (2.718281828) raised to any power in the range −88 to +88, the result always being positive.

Syntax: EXP (arithmetic expression). If the expression exceeds 88.0296919 when evaluated then an Overflow error is generated.

Errors: Syntax error – wrong command syntax e.g. missing closing bracket
Overflow error – expression exceeds 88.0296919

Use: EXP is the converse function of LOG and is used principally in scientific or statistical programs.

ROM routine entry point: $BFED

Routine operation: The value of e to the power of the value in FAC#1 is calculated. It first multiplies FAC#1 by a constant equal to 1/log e 2 which is then tested for range. If it is within the range then a series routine is called which calculates 2↑(x/log e 2). The result is stored in FAC#1.

FOR...TO...[STEP] and NEXT

Abbreviated entry: FOR is F(shift)O
TO has no abbreviation
STEP is ST(shift)E
NEXT is N(shift)E

Tokens:	FOR	Hex $81	Decimal 129
	TO	Hex $A4	Decimal 164
	STEP	Hex $A9	Decimal 169
	NEXT	Hex $82	Decimal 130

Modes: Direct and program

Purpose: This command is used to repeat the program contained in the lines between the FOR ... TO ... [STEP] command statement and its associated NEXT. With each repetition of the loop the variable is incremented by the STEP value until it reaches the value in the TO variable.

Syntax: FOR floating point variable = arithmetic expression or floating point variable TO arithmetic expression or floating point variable [STEP arithmetic expression or floating point variable]. The square brackets denote that the STEP command is optional; if STEP is not defined it defaults to a step increment of 1.

Errors: Syntax error – wrong command syntax e.g. integer or array variables used
NEXT without FOR – if there is no FOR ... TO to match a NEXT; this can occur if the NEXT is simply omitted or a RETURN is used with a GOSUB/GOTO, called before the FOR ... NEXT loop

Use: Although FOR ... NEXT loops are probably one of the most useful commands in Basic the version of Basic used in the CBM 64 has several interesting features which can pose problems for the programmer. The first problem likely to be encountered is with nested FOR ... NEXT loops. The level of nesting is limited by the fact that the processor stack is used to store the loop variables and takes 18 bytes of stack space for every nested loop. To ensure correct nesting it is advisable to omit the variable from the NEXT statement; this will ensure that the interpreter simply takes the last entered FOR ... TO entry on the stack as referring to the NEXT statement. The level of nesting is limited in theory to 10 levels, though in practice it is fewer since the stack is also required for other purposes.

This use of the stack also gives several other effects. When a new FOR ... TO is set up the stack is scanned for an existing active loop with the same variable. If found then the new FOR ... TO replaces the old one. A RETURN after a GOSUB also has the effect of clearing all stack contents placed there during the GOSUB routine, thus erasing any FOR ... TO references set up during the GOSUB which are still open (the cause of the NEXT without FOR error encountered in such cases). The only way in which the variable denoting the upper limit of the loop or the loop step can be changed is to directly change the value in the stack, since these two variables are stored as part of the stack data. Thus the variables used to define the upper limit and step can be reused immediately after the FOR ... TO ... STEP command is set up without affecting the command operation.

The STEP command and associated variable, if not specified, defaults to 1. It should be noted that the FOR ... NEXT command will always pass once through the loop. If STEP is specified it can lead to errors due to rounding of the floating point values; this will not occur with non fractional values except on very large numbers, but can be quite serious on some fractional values especially values like $\frac{1}{3}$. The result of such rounding can easily give a loop count error of plus 1, and is commonly encountered in routines like graphics circle drawing.

ROM routine entry point: FOR ... TO $A742
NEXT $AD1E

Routine operation: FOR ... TO setup evaluates the expression and then assigns 18 bytes on the stack for the active FOR loop, having checked that there is space on the stack. The format of the stack entry for an active FOR loop is:

Stack address	1	loop return address lo
	2	loop return address hi
	3	return line number hi
	4	return line number lo
	5	TO value in floating point notation (lsb)
	6	TO value in floating point notation (lsb)
	7	TO value in floating point notation (lsb)
	8	TO (most significant byte + sign)
	9	TO (mantissa)
	10	sign of STEP
	11	STEP value in floating point notation (lsb)
	12	STEP value in floating point notation (lsb)
	13	STEP value in floating point notation (lsb)
	14	STEP (most significant byte + sign)
	15	STEP (mantissa)
	16	variable address hi
	17	variable address lo
	18	FOR token $81

The first function of the NEXT routine is to check for any variable name following the NEXT command. If there is none then the locations $49 and $4A are set to zero. If a variable name follows the NEXT command then its location is obtained using the routine at $B08B. This returns the pointers in the accumulator (low order address byte) and the .y index register (high order address byte). These values are stored in the variable pointer $49,$4A. The stack is then searched for a matching FOR command. If no variable is specified then the last entered FOR return data is used; if there is no matching return FOR then a NEXT without FOR error is generated. The step value in floating point is moved from the stack to floating point accumulator #1 and added to the variable pointed to by $49,$4A. This is compared with the TO value stored on the stack, and if equal exits from the FOR ... NEXT loop. If not equal then the return line number is restored in $39,$3A and the charget pointers in $7A,$7B are reset to the FOR entry point and a warm start to Basic initiated to restart the program at that point.

FRE

Abbreviated entry: F(shift)R

Token: Hex $B8 Decimal 184

Modes: Direct and program

Purpose: Calculates the number of unused bytes of memory available between the bottom of the string storage area and the top of the array storage. The routine also performs a 'garbage collect' which clears all unused string variables out of memory thus freeing the maximum amount of available memory space.

Syntax: FRE (expression). Since FRE is a function it requires an expression. However, in the case of FRE this expression is purely a dummy and can be any value.

Errors: Syntax error – wrong command syntax

Use: This command is used principally in the direct mode to find the size of a program, or in the program mode, where a program involves a lot of string storage and manipulation, to prevent an Out of memory error being generated because of insufficient space to store a new string. String storage can quickly use up available memory if a lot of string manipulation is being performed. The reason is that every time a new string is created it is stored in the string storage area which starts at the top of available RAM memory and extends downwards until it meets the top of the array storage area. New strings are simply added to the bottom of this memory area, and when a string variable is redefined the old string is not erased; the variable pointers are simply changed to point to the new string. This means that the interpreter must occasionally remove unassigned strings in order to release more free memory. This process is called 'garbage collection' and occurs at irregular intervals whenever it is not possible to add another string or variable to memory.

Unfortunately garbage collection can be a very lengthy operation (it can be well in excess of 30 minutes) which totally halts the program operation. Many users have been faced with a machine which ceases to operate, and have come to the conclusion that it has crashed, when in reality it is simply performing a 'garbage collect'. It should be noted, of course, that the amount of memory available on the 64 means that a garbage collect situation is, on most programs, rarely ever reached. If it is thought likely to occur then there are two precautions which can be taken to reduce the garbage collection delay time. The first is to lower the top of memory using the top of memory pointers, to reduce the space available for string storage to the absolute minimum, thereby forcing frequent small garbage collects, each of fairly short duration. The other method is to use the FRE command to force a garbage collect at some regular period within the program where there is normally a pause in program operation, e.g. after an input prompt.

ROM routine entry point: $B37D

Routine operation: The routine discards all unwanted strings by calling the garbage collect routine and then calculates the amount of free memory available. This is returned as an integer stored as two bytes in $62 (lo), $63 (hi).

GET and GET#

Abbreviated entry: G(shift)E
G(shift)E#

Token: Hex $A1 Decimal 161

Modes: Program mode only

Purpose: These two commands input a single byte, GET from the keyboard and GET# from any other input device. If there are no characters in the keyboard buffer then these commands will simply return a null string.

Syntax: GET variable name, [variable name], [variable name], ...
GET # arithmetic expression, variable name, [variable name], ...
The GET and GET# commands may be followed optionally by more than one variable, but must always have at least one variable. The GET# command must always be followed by a logical file number between 1 and 255.

Errors: Syntax error – wrong command syntax, or attempting to input a non numeric character using GET numeric variable
Illegal direct error – attempt to use the commands in direct mode
Device not present – no input device corresponding to the logical file

Use: The GET commands have a great virtue over the INPUT commands in that they do not have the same conventions and restrictions. Therefore GET can be used to input any character (including " : , return and the screen editor characters which are not accepted in an INPUT command). The single character strings input by GET can then be validated by the program and if necessary concatenated to produce a longer string, thereby giving the programmer total control over input.

The GET command gets one character from the keyboard buffer where they are placed by the keyboard servicing part of the IRQ routine. The keyboard buffer is situated at $0277 and occupies 10 bytes of memory. This buffer is organised on a first in first out basis and the GET command takes a single character off the top of the buffer. If there are no characters in the buffer then GET will return a null character (it is for this reason that a GET command usually has to be structured as a loop which rejects all null characters and just returns the first character entered). Keypresses entered into the buffer prior to the GET command will be returned by GET instead of keys pressed during the GET command execution. This can be countered by clearing the keyboard buffer previously by setting the buffer pointer in $C6 to zero.

The GET# command is used primarily to get data, byte by byte, from either tape or disk, and as with GET its main value lies in the command's ability to take any character, such as colons and commas, rejected by the INPUT# command. When reading from tape the GET# command obtains characters from the cassette buffer. The cassette buffer is loaded with a 192 byte block of data from tape, the tape then pauses until this buffer is read, the pointers are reset and the next 192 byte block is read from tape (see *The Commodore 64 Kernal and Hardware Revealed* for further details on tape and disk storage).

ROM routine entry point: $AB7B

Routine operation: Checks are first made by the routine to determine the operation mode, direct or program, and whether the command is GET or GET#. If the command is in the direct mode it is rejected with an error message.

When it is a GET# command the file number is input, the routine checks that a comma is present and sets the required device for input. The input buffer at $0200 is then set up to accept just a single character, the buffer being filled with a null byte. The accumulator is loaded with $40 and the routine jumps to the GET character from the input device subroutine within the perform READ routine; the entry address is $AC0f. This routine first stores the accumulator in location $11 to identify that it is a GET command. The character is then obtained from the input device, the input character being stored in $0200.

GOSUB and RETURN

Abbreviated entry: GOSUB GO(shift)S
RETURN RE(shift)T

Tokens:		
GOSUB	Hex $8D	Decimal 141
RETURN	Hex $8E	Decimal 142

Modes: Direct and program

Purpose: This performs a jump to another section of the program specified by a line number following the GOSUB command. On encountering a RETURN command the program will then return to the instruction following the GOSUB. The section of program jumped to is called a subroutine.

Syntax: GOSUB line number. The line number must be in ASCII numerals and be within the range 0 to 63999. The RETURN command must be situated at the end of the subroutine called by GOSUB.

Errors: Syntax error – wrong command syntax e.g. line number out of range
Return without GOSUB – no GOSUB matching a RETURN
Undefined statement – line number does not exist
Out of memory – excessive use of GOSUB nesting using all the stack space

Use: This is a very important Basic command which allows the use of subroutines within a program, a subroutine being a piece of program code required more than once in a program. Like the loop command FOR ... NEXT, the pair of commands GOSUB ... RETURN make extensive use of the processor stack to save the return address and line number. Every time a GOSUB is used it requires eight bytes of the processor stack, therefore there is a limit to the number of levels to which subroutines can be nested within other subroutines. This limit is, in theory, 23 levels of nesting, but in practice it is much less since the stack is also required for other purposes such as FOR ... NEXT loops etc.

The RETURN command, when encountered, will delete all stack entries above and including the last entered GOSUB stack entry. Any attempt to use levels of GOSUB nesting greater than this will result in an Out of memory error. This error will also result if an attempt is made by a GOSUB to call itself (the stack will fill up with return addresses which are not deleted by the RETURN command).

It is sometimes useful to be able to escape from a GOSUB without executing a RETURN: this is done by the POP command in Chapter 5. Another interesting feature of the GOSUB command is that when it checks the line number following the GOSUB it performs an incomplete validation, thus GOSUB followed by no line number or a non numeric character will always default to a GOSUB 0, a potentially useful feature. However, any attempt to do a computed GOSUB will fail and will default to 0 if a variable is used, or to the number if used first. A proper computed GOSUB routine is given in Program 10.

Source code for computed GOSUB.

```
033C          !CALCULATED GOSUB
033C          !
C000          *=$C000
C000 20FDAE               JSR $AEFD                     !SCAN PAST COMMA
C003 208AAD               JSR $AD8A                     !GET LINE NUMBER
C006 20F7B7               JSR $B7F7                     ! INTO $14,$15
C009 68                   PLA                           !REMOVE SYS RETURN
C00A 68                   PLA                           ! ADDRESS
C00B A903                 LDA #$03
C00D 20FBA3               JSR $A3FB                     !CHECK STACK DEPTH
C010 A57B                 LDA $7B                       !PUSH OFF GOSUB
C012 48                   PHA                           !PARAMETERS
C013 A57A                 LDA $7A
C015 48                   PHA
C016 A53A                 LDA $3A
C018 48                   PHA
C019 A539                 LDA $39
C01B 48                   PHA
C01C A98D                 LDA #$8D
C01E 48                   PHA
C01F 20A3A8               JSR $A8A3                     !DO GOTO
C022 4CAEA7               JMP $A7AE                     !BACK TO MAIN LOOP
```

BASIC loader for computed GOSUB.

```
10 INPUT"ADDRESS FOR CALCULATED GOSUB";I:S=I
20 READA:IFA=-1THEN50
30 POKEI,A:I=I+1
40 T=T+A:GOTO20
50 IFT<>4560THENPRINT"▓▓CHECKSUM ERROR :4560"T:END
60 IFI<>S+37THENPRINT"▓▓NUMBER OF DATA ERROR":END
70 PRINT"▓▓TO USE THE CALCULATED GOSUB:"
80 PRINT"▓SYS("S"),LINE NUMBER"
90 END
100 DATA32,253,174,32,138,173,32
110 DATA247,183,104,104,169,3,32
120 DATA251,163,165,123,72,165,122
130 DATA72,165,58,72,165,57,72
140 DATA169,141,72,32,163,168,76
150 DATA174,167,-1
```

Program 10.

ROM routine entry point: GOSUB - $A883
RETURN - $A8D2

Routine operation: The routine to perform the GOSUB command pushes the seven bytes of data required for a GOSUB onto the stack, having first checked that there is space on the stack. If there is not then an Out of memory error is generated. The format of the stack entry for an active GOSUB is:

Stack address 1 $A7 return to control loop address msb
2 $E9 return to control loop address lsb
3 return address hi
4 return address to
5 line number lo
6 line number hi
7 $8D GOSUB token

Having placed this data on the stack the routine performs the same function as the GOTO command and scans the Basic program to locate the desired target line. It does this by first comparing the target line number with the current line number; if the target is larger then it scans up, if smaller than it scans up from the start of Basic. If the line is not found then an Undefined statement error is generated. Having found the line program the control jumps to it.

The routine to perform RETURN first checks for a GOSUB token $8D on the stack by calling routine $A38A. This searches for FOR entries on the stack which are then skipped and the next stack entry checked for a GOSUB. If found then all higher stack entries are erased and the pointers to the GOSUB calling routine recovered. If no GOSUB pointer is found then a RETURN without GOSUB error is generated. The original line number is stored in pointers $39,$3A. Charget is reset using the return address pointers from the stack. The routine then merges with the DATA routine which searches for the next statement after the pointer; this is used to ignore any commands following the GOSUB and to start execution on a new line following the GOSUB. The RTS terminating the DATA routine calls the routine pointed to by the return to control loop address on the stack.

GOTO

Abbreviated entry: G(shift)O

Token: Hex $89 Decimal 137

Modes: Direct and program

Purpose: Performs a jump to the specified line in the command. It can be used in conjunction with IF and ON to give conditional jumps.

Syntax: GOTO line number. The line number must be in ASCII numeric characters and in the range 0 to 63999.

Errors: Undefined statement – line number specified does not exist

Use: Programming purists do not approve of the GOTO command, however it is very useful especially for jumping on a conditional test. An interesting feature of the GOTO command is that if no line number is specified or a non numeric character follows the GOSUB, then the interpreter assumes a default of GOTO 0. Computed GOTOs are not allowed, but a simple routine to add this facility to Basic, is given in Program 11.

Source code for computed GOTO.

```
033C          !CALCULATED GOTO
033C          !
C000          *=$C000
C000 20FDAE              JSR $AEFD                     !SCAN PAST COMMA
C003 208AAD              JSR $AD8A                     !GET LINE NUMBER
C006 20F7B7              JSR $B7F7                     ! INTO $14,$15
C009 4CA3A8              JMP $A8A3                     !EXECUTE GOTO
```

BASIC loader for computed GOTO.

```
10 INPUT"ADDRESS FOR CALCULATED GOTO";I:S=I
20 READA:IFA=-1THEN50
30 POKEI,A:I=I+1
40 T=T+A:GOTO20
50 IFT<>1671THENPRINT"▓▓CHECKSUM ERROR :1671"T:END
60 IFI<>S+12THENPRINT"▓▓NUMBER OF DATA ERROR":END
70 PRINT"▓▓TO USE THE CALCULATED GOTO:"
80 PRINT"▓SYS("S"),LINE NUMBER"
90 END
100 DATA32,253,174,32,138,173,32
110 DATA247,183,76,163,168,-1
```

Program 11.

ROM routine entry point: $A8A0

Routine operation: The line number used in the GOTO is first fetched and stored in locations $14,$15. The line number is then compared with the current line number (note high bytes only are compared). If the target line# high byte is larger than the current line# high byte, then the program is scanned upwards from the current line using the link pointers to achieve this scanning quickly. If the target line number is not found then an Undefined statement error is generated. If the line is found then the address of the zero before the start of the target line is loaded into the charget pointers at $7A,$7B and program execution is restarted on an RTS.

IF...THEN

Abbreviated entry: IF None
THEN T(shift)H

Token: IF Hex $8B Decimal 139
THEN Hex $A7 Decimal 167

Mode: Direct and program

Purpose: This command allows the conditional execution of any statement following the IF including jumps or GOSUBs to other lines, depending on the value or expression following the IF statement. The IF command is usually associated with the THEN or GOTO commands.

Syntax: IF arithmetic or logical expression
THEN line number or expression
GOTO line number
THEN GOSUB line number
When GOTO is used it must not have a space between GO and TO.

Errors: Syntax error – wrong command syntax
Undefined statement – if the line number following THEN,GOTO or GOSUB does not exist

Use: The IF...THEN command structure is the primary conditional test in CBM 64 Basic and is therefore of great use. It functions by first evaluating the expression following the IF statement. If this gives a value greater or less than zero then the expression is deemed to be true; if the result is zero then the expression is false. If the expression is false then any further statements on the line are ignored and the next line executed. If the expression is true then the rest of the line – a THEN or GOTO statement – is executed plus any further commands separated by colons. If the IF command is followed not by an expression but just a variable, then the interpreter takes the value of the variable and uses that as the test. It should be noted, of course, that the sign of a value or expression is not considered by the IF command.

One interesting feature of the IF command is that the condition following the IF statement may be a string or string variable; it will not produce an error but will give some odd effects. If the condition is a string variable then the condition tests the contents of FAC#1 left after the previous numeric calculation or numeric variable assignment. A previous string assignment will also affect the condition test; a false condition will be generated only if the assignment was a null string. The IF conditional test will work satisfactorily when making comparisons between two strings. If a subscripted string variable is included within the test, then the interpreter will ignore the string variable and simply take the number or numeric variable used in the subscription as the test value. The use of a string as the conditional test variable will give a false message if it is a null string and a true message in all other cases. However, this does not clear the string stack, and using it three times will give a Formula too complex error.

ROM routine entry point: $A928

Routine operation: The expression following the IF is first evaluated by the routine at $AD9E, the result of the evaluated expression being placed in floating point accumulator #1. The exponent value is also placed in the processor accumulator. The routine then checks whether the following statement is the token for either THEN ($A7) or GOTO ($89); if not then a Syntax error is generated. When the result of the evaluation is zero the exponent in the accumulator is set to zero. If the accumulator contains a zero then the condition is deemed 'false' and the control branches to the next line. This is done by taking the scan offset to the next line start address in the .y index register and adding it to the charget pointers in $7A,$7B. If the accumulator is greater than zero then the condition is 'true' and the statement following the IF conditional expression is performed. A GOTO or THEN followed by a line number will execute a GOTO jump; if THEN is followed by GOSUB and a line number the GOSUB routine is executed. If THEN is followed by a variable then it is assigned.

INPUT and INPUT#

Abbreviated entry:	INPUT	None
	INPUT#	I(shift)N

Tokens:	INPUT	Hex $85	Decimal 133
	INPUT#	Hex $84	Decimal 132

Modes: Program mode only

Purpose: To input data into the computer, from the keyboard in the case of INPUT, and from a specified input device in the case of INPUT#. The INPUT command also displays the input on the screen at the current cursor position. An INPUT or INPUT# is terminated by the return key or a return ASCII character. INPUT can also include a string which is first output on the screen prior to data input.

Syntax: INPUT [string within quotes;] variable name [,variable name].....
INPUT# arithmetic expression, variable name [,variable name].....

With INPUT the string within quotes is optional as is also the use of more than one variable. When run, an INPUT command will first display any string following the command and then display a question mark followed by a flashing cursor as an input prompt. Extra input variables can be separated by commas. If carriage return is pressed after each then a double question mark is displayed to prompt. With the INPUT# command the arithmetic expression following defines the logical file number and must evaluate to a value between 1 and 255. It should be noted that there is no optional displayed string with the INPUT# command. With INPUT and INPUT# the maximum length of a data item input is 79 characters including the terminating 'return' character and question mark prompt.

Errors: Syntax error – wrong command syntax
Illegal direct – attempting to use INPUT commands in the direct mode
Redo from start – attempting to input the wrong variable type
Extra ignored – use of a comma separator within input
indicates that there are more inputs than variables
File not open – no input file open in INPUT#
Not input file – file not open for input

Use: Both INPUT and INPUT# have strict rules covering permissible input characters. The characters not accepted are principally , : " and the screen editor commands plus the 'return' character if used as anything other than an input terminator. These limitations can be quite annoying and are one reason why GET or GET# are often preferred to INPUT and INPUT# because these restrictions do not apply and the programmer can use his own character trapping. The INPUT routines treat the comma as a separator between inputs and therefore ignore it and place the following input data in the next variable (if one was assigned in the INPUT command; if not then an extra ignored error is produced). A colon encountered within the input data will signify the end of the statement. If the " character is input then all following characters are treated as being a literal string until a matching " is found.

The INPUT and INPUT# commands work by taking characters from the respective input device and placing them in the input buffer. This is an 88 byte block of memory at locations $0200 to $0257. Characters continue to be put in this buffer until either the buffer contains 80 characters (in which case a String too long error is generated) or a carriage return, comma or colon character is input. When a carriage return or separator character is input a terminating zero is added to the end of the input, and the buffer contents are assigned to the designated variable.

ROM routine entry point: INPUT – $ABBF
INPUT# – $ABA5

Routine operation: The INPUT routine first checks for a quotation mark, $22, as the next character following the INPUT command. If a quotation mark is present then the string within the quotation marks is printed on the output device. The input buffer at $0200 is set up to accept up to 80 characters, the status ST is then tested (derived from the value in $13), and the routine branches to the input line routine. It is the input line routine which is the cause of INPUT not accepting colons etc.

The INPUT# command gets the file number and checks for a following comma, sets the input device and jumps into the main input routine. Having performed the input, the input device is turned off and location $13 set to zero.

INT

Abbreviated entry: None

Token: Hex $B5 Decimal 181

Modes: Direct and program

Purpose: Converts the value in the argument into an integer by removing the fractional component of the value.

Syntax: INT (arithmetic expression). The arithmetic expression must be given a valid numeric result within the range acceptable for floating point values.

Errors: None

Use: The INT command is principally used in rounding values to whole numbers. However, since it removes just the fractional component of the number it will always round down all positive values and round up all negative values. To round up simply add .5 to the value then do an INT command. It should be noted that the value returned by the INT command is a floating point value and should not be confused with numbers stored as integers which have a maximum range of +32767 to −32768.

ROM routine entry point: $BCCC

Routine operation: This takes a value stored in FAC#1 and rounds it down to the nearest integer which is left in full floating point form in FAC#1.

LEFT$

Abbreviated entry: LE(shift)T

Token: Hex $C8 Decimal 200

Modes: Direct and program

Purpose: This takes the specified string and takes from it a substring consisting of the specified number of characters at the left end of the string.

Syntax: LEFT$(string expression, arithmetic expression). The string expression can be a string literal, string variable, a string function like LEFT$, or a combination of one or all of these, the only limitation being that the resulting string length must not exceed 255 characters. The arithmetic expression must be an integer number between 0 and 255 when evaluated.

Errors: Illegal quantity – value exceeds the limits 0 to 255

Use: The string functions are extensively used to manipulate strings and LEFT$. The principal use is in getting rid of trailing characters or truncating strings to a fixed length.

ROM routine entry point: $B700

Routine operation: The string parameter data is first pulled from the stack by the routine at $B761. The .y index register contains the string length. The bulk of the routine from $B706 is shared with MID$ and RIGHT$ and involves creating a substring, storing it in memory and setting up the necessary pointers.

LEN

Abbreviated entry: None

Token: Hex $C3 Decimal 195

Modes: Direct and program

Purpose: Will return the length of a string or string expression.

Syntax: LEN (string expression). The string expression must be valid and can be either a string variable, string literal, or string function. The combined string length must not exceed 255 characters.

Errors: Type mismatch - if it is a non string expression

Use: LEN is often used within FOR...NEXT loops to perform an operation on each character in a string.

ROM routine entry point: $B77C

Routine operation: This calls the routine at $B782 to obtain the string length which is returned in the accumulator with the.y index register set to zero. It then jumps to the routine at $B3A7 which converts the contents of .a and .y to a floating point value in FAC#1.

LET

Abbreviated entry: L(shift)E or by default nothing

Token: Hex $88 Decimal 136

Modes: Direct and program

Purpose: To assign a value or string to a variable

Syntax: LET is not actually required in CBM 64 Basic since if the first byte in any statement is not a token then the interpreter assumes that a LET command is intended by default. The interpreter parser then checks for an = sign following the variable and an expression or value following the equals sign. The type of variable allowed by the assignment is determined by the variable name. If the last character is % then the variable is an integer variable, if $ then it is a string variable; in all other cases a floating point variable is assumed. Variables can be either simple or array variables.

Errors: Type mismatch - wrong variable type assignment
Illegal quantity - value is outside the permitted size range

Use: The LET command is not necessary in CBM 64 Basic. For further details of variable type, storage and assignment see Chapter 2.

ROM routine entry point: $A9A5

Routine operation: The variable defined in the LET statement is first searched for amongst existing Basic variables using the routine at $$B08B. If it does not yet exist then it is set up. The variable pointer address is stored in locations $49,$4A. The routine then checks for an = sign (character value $B2). If this is not found then a Syntax error is generated. The value, string or expression following the equals sign is then evaluated and assigned to the corresponding variable pointed to by locations $49,$4A. The following are the start of the routines which assign the different variable types:

$A9C4	assign integer variables
$A9D6	assign floating point variables
$AA2C	assign strings, except
$A9D9	which assigns TI$

The routine which assigns TI$ uses a routine at $AA1D, which adds an ASCII digit to the contents of FAC#1. The digit is pointed to by $22,.y.

LIST

Abbreviated entry: L(shift)I

Token: Hex $9B Decimal 155

Modes: Direct and program mode. In program mode this command will stop the program after listing the desired lines.

Purpose: This command will output all or part of the Basic program stored in memory on the current output device.

Syntax: LIST [line number][-[line number]]. The beginning and end of line numbers defining the listing range are optional; the line numbers specified need not actually exist but must be within the range 0 to 63999.

Errors: Syntax error - wrong command syntax or if an unrecognisable token is encountered when listing

Use: The LIST command converts the tokenised Basic program back into an easily readable format which is displayed on the screen or to another peripheral (if the OPEN and CMD commands have previously been used to set up an output device); this would normally be a printer. The program can also be listed to cassette, disk or via a modem; this is often useful when transferring programs to non CBM devices. Another use for a Basic program listed to cassette is a simple merge routine. This is dealt with in *The Commodore 64 Kernal and Hardware Revealed.*

The LIST command has one quirk. After a REM command all shifted characters will be interpreted as tokens and output in their expanded form unless the shifted characters are enclosed in quotes. This can be utilised when listing the REM command by including screen or printer control characters after the REM, thereby either improving the listing's legibility or providing a degree of unlistability by using cursor characters to backspace over lines and thereby hide their contents.

A very useful variation of the LIST command is given in Program 12; it is modified to convert all the graphics screen control and colour characters into more readable form.

```
1 RESTORE
5 GOTO3000
10 DATA162,0,165,43,133,251,165
20 DATA44,133,252,160,0,177,251
30 DATA133,253,200,177,251,133,254
40 DATA201,0,208,1,96,200,200
50 DATA200,177,251,201,0,208,13
60 DATA165,253,133,251,165,254,133
70 DATA252,162,0,76,10,192,201
80 DATA34,208,10,232,224,2,208
90 DATA12,162,0,76,28,192,224
100 DATA1,240,3,76,28,192,201
110 DATA255,208,3,76,28,192,133
120 DATA102,24,201,192,144,4,216
130 DATA56,233,96,201,96,176,7
140 DATA201,33,144,3,76,28,192
150 DATA134,100,133,101,132,92,162
160 DATA1,200,177,251,197,102,208
170 DATA4,232,76,106,192,134,102
180 DATA224,1,240,2,176,10,202
190 DATA165,101,201,32,208,3,76
200 DATA167,193,138,133,93,169,10
210 DATA133,94,162,8,169,0,6
220 DATA93,42,197,94,144,4,229
230 DATA94,230,93,202,208,242,216
240 DATA24,105,48,133,94,24,216
250 DATA165,93,105,48,133,93,165
260 DATA101,201,97,176,3,76,22
270 DATA193,201,123,144,3,76,22
280 DATA193,216,56,233,32,133,101
290 DATA162,7,165,93,201,48,208
300 DATA8,202,165,94,201,48,208
310 DATA1,202,228,102,240,11,176
320 DATA6,32,0,194,76,227,192
330 DATA32,168,194,164,92,169,91
340 DATA145,251,165,93,201,48,240
350 DATA3,200,145,251,165,102,201
360 DATA1,240,5,165,94,200,145
370 DATA251,169,71,200,145,251,169
380 DATA62,200,145,251,165,101,200
390 DATA145,251,169,93,200,145,251
400 DATA166,100,76,28,192,133,101
410 DATA169,80,133,98,169,195,133
```

```
420 DATA99,162,80,160,0,177,98
430 DATA197,101,240,9,200,200,200
440 DATA202,16,244,76,250,198,200
450 DATA177,98,133,193,200,177,98
460 DATA133,99,165,193,133,98,160
470 DATA0,177,98,133,193,216,24
480 DATA105,4,170,165,93,201,48
490 DATA208,8,202,165,94,201,48
500 DATA208,1,202,228,102,240,11
510 DATA176,6,32,0,194,76,105
520 DATA193,32,168,194,164,92,169
530 DATA91,145,251,165,93,201,48
540 DATA240,3,200,145,251,165,102
550 DATA201,1,240,5,165,94,200
560 DATA145,251,132,92,160,0,234
570 DATA234,200,177,98,132,194,164
580 DATA92,200,145,251,132,92,164
590 DATA194,196,193,208,238,164,92
600 DATA169,93,200,145,251,166,100
610 DATA76,28,192,164,92,166,100
620 DATA76,28,192,-1
630 DATA134,194,165,102,56,229,194
640 DATA133,187,24,165,251,101,92
650 DATA133,95,165,252,105,0,133
660 DATA96,165,95,101,187,133,90
670 DATA165,96,105,0,133,91,165
680 DATA45,56,229,90,133,88,168
690 DATA165,46,229,91,170,232,152
700 DATA240,31,165,90,24,101,88
710 DATA133,90,144,3,230,91,24
720 DATA165,95,101,88,133,95,144
730 DATA2,230,96,152,73,255,168
740 DATA200,198,91,198,96,177,90
750 DATA145,95,200,208,249,230,91
760 DATA230,96,202,208,242,56,165
770 DATA45,229,187,133,45,176,3
780 DATA198,46,56,160,0,165,253
790 DATA229,187,133,253,145,251,133
800 DATA87,165,254,233,0,200,133
810 DATA254,133,88,145,251,136,177
820 DATA87,133,185,200,177,87,133
830 DATA186,240,24,136,56,165,185
840 DATA229,187,170,145,87,165,186
850 DATA233,0,200,145,87,133,88
860 DATA138,133,87,76,131,194,96
870 DATA138,56,229,102,133,187,24
880 DATA165,92,101,187,176,4,201
890 DATA254,144,3,76,65,199,165
900 DATA45,101,187,170,165,46,105
910 DATA0,197,56,208,7,228,55
920 DATA144,3,76,99,199,24,165
930 DATA45,133,90,101,187,133,88
940 DATA165,46,133,91,105,0,133
950 DATA89,165,251,101,92,133,95
960 DATA165,252,105,0,133,96,32
970 DATA191,163,24,160,0,165,45
980 DATA101,187,133,45,144,3,230
990 DATA46,24,165,253,101,187,133
1000 DATA253,133,87,145,251,165,254
1010 DATA105,0,200,133,254,133,88
1020 DATA145,251,136,177,87,133,185
1030 DATA200,177,87,133,186,240,24
1040 DATA136,24,165,185,101,187,170
1050 DATA145,87,165,186,105,0,200
1060 DATA145,87,133,88,138,133,87
1070 DATA76,19,195,96,-1
1080 DATA5,56,197,17,60,197,18
```

```
1090 DATA63,197,19,67,197,28,71
1100 DATA197,29,75,197,30,78,197
1110 DATA31,82,197,32,86,197,96
1120 DATA90,197,123,94,197,124,98
1130 DATA197,125,102,197,126,106,197
1140 DATA127,108,197,129,112,197,133
1150 DATA116,197,134,119,197,135,122
1160 DATA197,136,125,197,137,128,197
1170 DATA138,131,197,139,134,197,140
1180 DATA137,197,144,140,197,145,144
1190 DATA197,146,147,197,147,151,197
1200 DATA148,155,197,149,159,197,150
1210 DATA163,197,151,169,197,152,173
1220 DATA197,153,177,197,154,183,197
1230 DATA155,189,197,156,193,197,157
1240 DATA197,197,158,200,197,159,204
1250 DATA197,160,208,197,161,214,197
1260 DATA162,218,197,163,222,197,164
1270 DATA226,197,165,230,197,166,234
1280 DATA197,167,238,197,168,242,197
1290 DATA169,246,197,170,250,197,171
1300 DATA254,197,172,2,198,173,6
1310 DATA198,174,10,198,175,14,198
1320 DATA176,18,198,177,22,198,178
1330 DATA26,198,179,30,198,180,34
1340 DATA198,181,38,198,182,42,198
1350 DATA183,46,198,184,50,198,185
1360 DATA54,198,186,58,198,187,62
1370 DATA198,188,66,198,189,70,198
1380 DATA190,74,198,191,78,198,1
1390 DATA82,198,2,88,198,8,94
1400 DATA198,9,100,198,14,106,198
1410 DATA142,112,198,141,118,198,-1
1420 DATA3,87,72,84,2,67,68
1430 DATA3,82,69,86,3,72,79
1440 DATA77,3,82,69,68,2,67
1450 DATA82,3,71,82,78,3,66
1460 DATA76,85,3,83,80,67,3
1470 DATA71,62,42,3,71,62,43
1480 DATA3,71,60,45,3,71,62
1490 DATA45,1,126,3,71,60,42
1500 DATA3,79,82,71,2,70,49
1510 DATA2,70,51,2,70,53,2
1520 DATA70,55,2,70,50,2,70
1530 DATA52,2,70,54,2,70,56
1540 DATA3,66,76,75,2,67,85
1550 DATA3,79,70,70,3,67,76
1560 DATA83,3,68,69,70,3,66
1570 DATA82,78,5,76,32,82,69
1580 DATA68,3,71,82,49,3,71
1590 DATA82,50,5,76,32,71,82
1600 DATA78,5,76,32,66,76,85
1610 DATA3,71,82,51,3,80,85
1620 DATA82,2,67,76,3,89,69
1630 DATA76,3,67,89,78,5,71
1640 DATA62,83,80,67,3,71,60
1650 DATA75,3,71,60,73,3,71
1660 DATA60,84,3,71,60,64,3
1670 DATA71,60,71,3,71,60,43
1680 DATA3,71,60,77,3,71,60
1690 DATA92,3,71,62,92,3,71
1700 DATA60,78,3,71,60,81,3
1710 DATA71,60,68,3,71,60,90
1720 DATA3,71,60,83,3,71,60
1730 DATA80,3,71,60,65,3,71
1740 DATA60,69,3,71,60,82,3
1750 DATA71,60,87,3,71,60,72
```

```
1760 DATA3,71,60,74,3,71,60
1770 DATA76,3,71,60,89,3,71
1780 DATA60,85,3,71,60,79,3
1790 DATA71,62,64,3,71,60,70
1800 DATA3,71,60,67,3,71,60
1810 DATA88,3,71,60,86,3,71
1820 DATA60,66,5,67,84,82,76
1830 DATA65,5,67,84,82,76,66
1840 DATA5,67,84,82,76,72,5
1850 DATA67,84,82,76,73,5,67
1860 DATA84,82,76,78,5,67,82
1870 DATA71,62,78,5,67,82,71
1880 DATA62,77,-1
1890 DATA165,95,201,27,144,3,76
1900 DATA105,199,105,64,141,24,199
1910 DATA169,19,133,98,169,199,133
1920 DATA99,76,65,193,5,67,84
1930 DATA82,76,74,32,78,73,32
1940 DATA83,82,65,72,67,32,89
1950 DATA78,65,77,32,79,79,84
1960 DATA32,13,32,78,73,32,78
1970 DATA87,79,78,75,32,84,79
1980 DATA78,32,82,65,72,67,32
1990 DATA13,162,20,189,24,199,32
2000 DATA210,255,202,208,247,160,2
2010 DATA177,251,133,57,200,177,251
2020 DATA133,58,32,201,189,104,104
2030 DATA76,25,192,234,234,234,234
2040 DATA32,53,164,-1
3000 A=49152:B=49581:C=49664:D=49975:E=50000:F=50236:G=50488:H=50811
3010 I=50938:J=51045:K=0
3015 PRINT"▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓ LISTER PROGRAM "
3020 PRINT"▓▓▓▓▓ PROCESSING DATA - PLEASE WAIT "
3030 FORZ=ATOB:READY:IFY=-1THEN3060
3035 K=K+Y:POKEZ,Y:NEXT
3040 READY:IFY=-1THEN4000
3050 IFK=58105THEN4000
3060 PRINT"▓▓▓ DATA ERROR IN LINES 10 - 620 ":END
4000 K=0:FORZ=CTOD:READY:IFY=-1THEN4030
4005 K=K+Y:POKEZ,Y:NEXT
4010 READY:IFY=-1THEN5000
4020 IFK=41386THEN5000
4030 PRINT"▓▓▓ DATA ERROR IN LINES 630 - 1070 ":END
5000 K=0:FORZ=ETOF:READY:IFY=-1THEN5030
5005 K=K+Y:POKEZ,Y:NEXT
5010 READY:IFY=-1THEN6000
5020 IFK=35484THEN6000
5030 PRINT"▓▓▓ DATA ERROR IN LINES 1080 - 1410 ":END
6000 K=0:FORZ=GTOH:READY:IFY=-1THEN6030
6005 K=K+Y:POKEZ,Y:NEXT
6010 READY:IFY=-1THEN7000
6020 IFK=17491THEN7000
6030 PRINT"▓▓▓ DATA ERROR IN LINES 1420 - 1880 ":END
7000 K=0:FORZ=ITOJ:READY:IFY=-1THEN7030
7005 K=K+Y:POKEZ,Y:NEXT
7010 READY:IFY=-1THEN8000
7020 IFK=11236THEN8000
7030 PRINT"▓▓▓ DATA ERROR IN LINES 1890 - 2040 ":END
8000 PRINT"▓▓▓▓▓▓ DATA HAS BEEN INPUT "
8010 PRINT"▓▓▓▓▓ SYS 49152 TO USE "
8030 END
```

Program 12.

ROM routine entry point: $A69C

Routine operation: The routine first checks and sets up parameters, converting the line number from floating point into a memory address for the start of the link address of the Basic line in memory. The start address of the lowest line number is stored in locations $5F,$60 and the highest line number in $14,$15. If no parameters are given in the command then the lowest start address defaults to $0801 and the highest to $FFFF. Two important and useful routines are used: $A6C9 lists a line of Basic pointed to by $14,$15 to the output device, and $A717 converts a token value stored in the accumulator into a Basic keyword. The LIST routine involves two loops. The outer loop tests for the STOP key then prints carriage return and compares the next line number with the upper limit line number; if smaller it then prints the next line number. The inner loop displays the line character by character. It checks for a quote character, zero, and characters with ASCII codes greater than 128. If it finds a quote then all following characters are printed exactly as stored until another quote is found. An ASCII character is interpreted as a token and is expanded, the full expanded form being printed. A zero indicates that the line has terminated and the inner loop is closed, the outer loop being called again.

LOAD

Abbreviated entry: L(shift)O

Token: Hex $93 Decimal 147

Modes: Direct and program

Purpose: To retrieve a program or memory dump from a storage device back into RAM memory, storage devices being either disk or tape.

Syntax: LOAD [string expression [, arithmetic expression [, arithmetic expression]]]. All the parameters within square brackets are optional. The string expression is the name of the program to be loaded; if omitted then the first program encountered is loaded. When used with a disk drive the program name must always be used. The first arithmetic expression is the device number which is one for the tape drive and eight for disk on the Commodore 64. The second arithmetic expression always follows the first and defines where the program will start in memory. If this value is zero, or no value is used, then the program will always start loading at an address pointed to by the contents of the .x and the .y index registers. This is normally the start of the Basic program storage area in the normal mode of operation. If the second arithmetic expression is <>0 then the program will start loading at the address from which it was saved. The secondary address will have no effect on loading from tape if a secondary address of three was used in the SAVE command.

Errors: Load error – when verifying a procedure this indicates an error in the loaded program

Device not present – specified device is not connected

Missing file name – no file name was specified when loading from disk
Break error – if run/stop key is pressed
Illegal device number – invalid device number
Illegal quantity – out of range device or secondary address values (range is 1 to 255)

Use: The functioning of this command varies according to whether it is used in direct or program mode. In direct mode the computer produces a series of messages which are displayed on the screen. These are:

```
Disk - LOAD "PROGRAM",8
       SEARCHING FOR PROGRAM
       LOADING
       READY

Tape - LOAD["PROGRAM 2"[, 1[,0]]]
       PRESS PLAY ON TAPE
       SEARCHING [FOR PROGRAM 2]
       [FOUND PROGRAM 1]
       FOUND [PROGRAM 2]
       LOADING [PROGRAM 2]
       READY
```

On tape the square brackets denote that if the program name is not included in the LOAD command then it will not be displayed in the messages, and the first program encountered on tape will be loaded. If the name is specified, and that program is not the first on tape, then the name of each program encountered will be displayed. Of course, if the program is not found then a File not found error will also be displayed.

In program mode the only message displayed by the LOAD command is PRESS PLAY ON TAPE when loading from tape. The program will load correctly, replacing the existing program and will start running from the beginning of the new program as soon as the loading is completed. There is one problem with using LOAD in the program mode; it does not change the variable pointers of the old program. This means that if the new program is larger than the old, it will be impossible to pass variables between the two programs, and because the variable pointers have not been set correctly for the new program, a crash will occur as soon as one tries to assign a variable. The best way to guard against this is to make sure that the start of the variable pointers is always set to an address above the end of the longest of the chained programs, thereby ensuring that variables will never be overwritten by a program. The setting of variable pointers can be achieved by finding the longest program and getting its start of variable pointer by peeking locations 45 and 46. These values should then be poked into these same two locations as the very first command of the first program in the chain.

The method of loading and running the first program on tape or disk is by pressing the SHIFT/RUN keys. This then works by forcing the command LOAD and RUN into the keyboard buffer. The interpreter then executes these as two direct mode commands.

ROM routine entry point: $E168

Routine operation: This routine loads a program into the computer from disk or tape. After loading, if an error has occurred, the error message is printed, otherwise a check on direct mode is made. If in direct mode the variable pointers are set to the end of the program. READY is output and a CLR performed. If in program mode then charget is reset to the beginning of the program, the program is re-chained and the Basic program executed.

LOG

Abbreviated entry: None

Token: Hex $BC Decimal 188

Modes: Direct and program

Purpose: Calculates the logarithm to the base e of any positive non zero arithmetic expression.

Syntax: LOG(arithmetic expression). The arithmetic expression must be a positive non zero value within the permissible limits of a floating point number.

Errors: Illegal quantity - the arithmetic expression has a zero or negative value

Use: LOG is the converse function of EXP and is used principally in scientific or statistical programs.

ROM routine entry point: $B9EA

Routine operation: This calculates the logarithm to the base e of a value stored in FAC#1 and puts the result in FAC#1. The logarithm is calculated using a fairly complex series evaluation.

MID$

Abbreviated entry: M(shift)I

Token: Hex $CA Decimal 202

Modes: Direct and program

Purpose: This takes the specified string and takes from it a substring.

Syntax: MID$(string expression, arithmetic expression [,arithmetic expression]). The string expression can be either a string literal, string variable, a string function like LEFT$, or a combination of one or all of these concatenated with the + sign, the only limitation being that the resulting string must not be longer than 255 characters. The arithmetic expressions, which must be within the range 0 to 255, define the starting and ending character positions of the substring within the main string. If the second arithmetic expression is omitted then the substring will continue to the end of the main string. This has a similar function to RIGHT$ but is often more useful, since it does not take just the designated number of characters from the right of the string, but starts at a designated character position within the string and takes all characters to the right irrespective of how many there are.

Errors: Illegal quantity – if either of the arithmetic expressions exceeds the permissible range 0 to 255

Use: The string functions are used extensively to manipulate strings and MID$. The principal use is in splitting up long strings.

ROM routine entry point: $B737

Routine operation: This checks the syntax and pulls the parameters from the stack before jumping into the LEFT$ routine at $B70E. This creates the substring, stores it in memory and sets up the necessary pointers.

NEW

Abbreviated entry: None

Token: Hex $A2 Decimal 162

Modes: Direct and program mode

Purpose: This command erases a Basic program in memory by erasing the link address to the first line. The program can be resurrected after NEW with the OLD command in Chapter 5.

Syntax: NEW. There are no parameters.

Errors: Syntax error – if the character following the NEW token is neither a colon nor end of line, or if the first byte of the Basic program storage area does not contain a zero

Use: This command erases the program in memory by putting zeros into locations $801 and $802 (assuming the normal start of a Basic address). This means that virtually all other memory locations are unaltered, therefore NEW will have no effect on machine code programs (such programs should, of course, not start at the beginning of the Basic area).

ROM routine entry point: $A642

Routine operation: This places zero into the first two bytes of the Basic RAM program storage area. The end of Basic pointers $2D,$2E are then loaded with the address of the start of the Basic storage area +2 bytes. Finally the routine at $A68E sets the charget pointers $7A, $7B to point to the start of Basic storage −1. The CLR routine is then entered.

NOT

Abbreviated entry: N(shift)O

Token: Hex $A8   Decimal 168

Modes: Direct and program

Purpose: This will evaluate the complement of the arithmetic expression following the command.

Syntax: NOT arithmetic or logical expression. The arithmetic or logical expression must be within the range +32767 to −32768 when evaluated.

Errors: Illegal quantity - values outside the range +32767 to −32768

Use: It should be noted that this command operates on the binary value not the decimal value and performs a twos complement on the binary value. This has the effect of converting all binary ones into zeros and vice versa. The NOT command has the highest priority in the hierarchy of logical operators and thus takes precedence over AND and OR.

ROM routine entry point: $AED4

Routine operation: This converts the evaluated expression in FAC#1 into integer format and performs a NOT operation on locations $64,$65. It then refloats the value into FAC#1.

ON

Abbreviated entry: None

Token: Hex $91   Decimal 145

Modes: Direct and program

Purpose: This command is always linked with either GOTO or GOSUB and causes a branch to one of a series of line numbers; which one is dependent on the value of the variable following ON.

Syntax: ON arithmetic expression GOTO line number, line number, ...
ON arithmetic expression GOSUB line number, line number, ...
The expression following the ON command must evaluate to a number in the range 0 to 255. (*Note:* The value will always be rounded down to an integer.)

Errors: Illegal quantity - if the arithmetic expression is outside the range 0 to 255
Syntax error - if the wrong command syntax is used e.g. if GOTO or GOSUB does not follow an arithmetic expression or if there is no space between GO and TO

Use: This is a multiple exit conditional branch, thus when the value of the expression is 1 then the branch is to the first line number, if 2 then the second line number and so on. When the value exceeds the number of line numbers specified after the GOSUB or GOTO command then the program simply branches to the following line. If no line number is specified then a default of line 0 is assumed by the interpreter. Thus a line ON X GOTO ,,,,20,,,50 is valid and just means that if .x is 1,2,3,4,6 or 7 then the control will branch to line 0.

ROM routine entry point: $A94B

Routine operation: This checks the variable type and evaluates it using the routine at $B79E, which returns the value in location $65 and the .x index register. It then checks whether the next command following the ON is a token for either GOTO ($89) or GOSUB ($8D); if it is neither of these a Syntax error is generated. It then goes through a loop which decrements the value in location $65, gets the first line number from the list of line numbers following the GOTO or GOSUB command, and checks for a comma following it. This loop is then repeated, decrementing $65 and getting the next line number and so on until either the value in $65 is zero or the line numbers are exhausted. If the contents of $65 are zero then the next line number to be accessed is the line to which the program control will be transferred. If the contents of $65 are not zero and there are no more line numbers then the next statement is executed by default.

OPEN

Abbreviated entry: O(shift)P

Token: Hex $9F Decimal 159

Modes: Direct and program

Purpose: This statement opens an I/O channel for input and/or output to a peripheral device.

Syntax: OPEN arithmetic expression [,arithmetic expression [,arithmetic expression [,string expression]]]. The first expression is the logical file number and is compulsory; it must evaluate to a number within the range 1 to 255. The second arithmetic expression is the device number. This is hardware specific (thus disk drives are usually device 8) and must be a value between 0 and 15. The third arithmetic expression is a secondary address which is also hardware dependent and is used to send commands to the peripheral. The final string expression is the file name; it can also include file type and mode designators.

Errors: Device not present – device corresponding to the device number is not attached
File open – file has already been opened
Too many files – more than 10 files are already open
Illegal device number – device number is outside range
Out of memory – RS232 channel has insufficient memory for buffers

Use: The open command is an essential part of the Basic file handling commands. The full functioning and operation of this command is dealt with in *The Commodore 64 Kernal and Hardware Revealed* and *The Commodore 64 Disk Drive Revealed* in this series.

ROM routine entry point: $E1BE

Routine operation: This routine opens a logical file on a specified device for reading or writing. It first gets the parameters using routine $E1D6 and then uses the kernal routine at $FFC0 to open the file (see *The Commodore 64 Kernal and Hardware Revealed* for further information on these routines).

OR

Abbreviated entry: None

Token: Hex $B0 Decimal 176

Modes: Direct and program

Purpose: This command performs a logical OR between two expressions. These expressions are first converted into double byte integer values, an OR performed and the result returned as a two byte integer.

Syntax: expression A OR expression B. The expression can be either arithmetic or logical but must always be either an integer value or a floating point value within the range +32767 and −32768.

Errors: Syntax error – incorrect command syntax
Illegal quantity – if the expressions exceed maximum/minimum values
Type mismatch – using a non arithmetic or logical expression

Use: The OR command acts either as a logical operator or as a bitwise operator on two straight sixteen bit values.

As a logical operator the OR command is used to ensure that at least one of two conditions is met before a particular operation is performed, as in the following example:

```
IF A<2 OR A>8 THEN PRINT "VALUE IN RANGE"
```

The result of a comparison gives −1 if the comparison is true and 0 if it is false. If a comparison is true then a value of −1 is returned by the comparison routine. This is represented as a twos complement value with a binary representation of:

1111 1111 1111 1111 or hex $FFFF or −1

Similarly a false comparison returns a value of zero, represented as:

0000 0000 0000 0000 or hex $0000 or 0

Therefore an OR will give a true condition only when one or both conditions are true (both values are $FFFF); all other states will be regarded as false.

A bitwise OR compares the first bit of one value with the first bit of the second value and gives a result according to the following truth table:

OR	1	0
1	1	1
0	1	0

Thus the command:

```
1278 OR 3279
```

has as its binary equivalent:

```
     0000  0100  1111  1110
OR   0000  1100  1100  1111
```

This gives the result:

```
0000  1100  1111  1111
```

or decimal 3327.

It should be noted, of course, that the OR operation is performed on two signed double byte integers, which are stored in twos complement form. Thus a value of −1 has a binary equivalent of 1111 1111 1111 1111 and any number ORed with −1 will always return the same number. Likewise a positive value ORed with a negative will always give a negative result.

The hierarchy of logical operators is NOT, AND, OR, thus NOT and AND always have a higher priority than OR.

ROM routine entry point: $AFE6

Routine operation: The two arguments in floating point format are stored in FAC#1 and FAC#2. They are first converted to fixed point integer values, the

OR operation performed on the two sixteen bit numbers, and the result converted back from integer to floating point in FAC#1.

PEEK

Abbreviated entry: P(shift)E

Token: Hex $C2 Decimal 194

Modes: Direct and program mode

Purpose: This command gets the contents of a desired memory location and returns its decimal value in the designated variable.

Syntax: PEEK (arithmetic expression). The arithmetic expression must be positive and all non integer values will be integerised; the value must be within the range 0 to 65535.

Errors: Illegal quantity – value is negative or outside the range 0 to 65535

Use: This command is invaluable in any application which requires direct access to memory locations. The principal applications are in passing parameters between machine code routines and Basic, manipulating screen displays, using the VIC, I/O and SID chips and manipulating Basic variables. It should be noted that the only locations which cannot be PEEKed are $14,$15, the reason being that these two locations contain the variable used by PEEK. For a double byte version of this command see the DEEK command in Chapter 5.

ROM routine entry point: $B80D

Routine operation: The memory address parameter has previously been obtained using routine $B7F7. The parameter is thus stored as a two byte integer in locations $14,$15. The result is put in the .y index register. This is then converted to floating point form in FAC#1 by the routine $B3A2.

POKE

Abbreviated entry: P(shift)O

Token: Hex $97 Decimal 151

Modes: Direct and program mode

Purpose: This command puts the contents of a designated variable into a desired memory location.

Syntax: POKE (arithmetic expression) (arithmetic expression). The first arithmetic expression defines the desired memory location and must be positive; all non integer values will be integerised. The value must be within the range 0 to 65535. The second expression is the value to be placed in the memory location; this must be a positive value in the range 0 to 255. Attempts to POKE data to a ROM memory location will produce no effect on the ROM but will place the data in the corresponding RAM memory plane.

Errors: Illegal quantity – value is negative or outside the range 0 to 65535

Use: This command is invaluable in any application which requires direct access to memory locations. The principal applications are in passing parameters between machine code routines and Basic, manipulating screen displays, using the VIC, I/O and SID chips and manipulating Basic variables. One use of the POKE command is to transfer the ROM based operating system and Basic software to the corresponding RAM memory plane by using a PEEK followed by a POKE to the same locations in ROM. For a double byte version of this command see DOKE in Chapter 5.

ROM routine entry point: $B824

Routine operation: The memory address parameter and contents parameter are obtained using routine $B7EB. This leaves the address parameter in $14,$15 and the value parameter in the .x index register. This value is then transferred to the accumulator and stored in memory at the address pointed to by the first parameter in $14,$15.

POS

Abbreviated entry: None

Token: Hex $B9 Decimal 185

Modes: Direct and program

Purpose: It returns the position of the cursor on the current screen line. It should be noted that although the CBM 64 has only a 40 column screen, it works on an 80 character line by folding each output line onto two lines. Therefore if the POS command returns a value between 40 and 79 then it is located on the second display line.

Syntax: POS(expression). The expression used by the POS function is a dummy variable and any numerical expression is valid.

Errors: None

Use: This command has fairly limited applications. These are limited to tests for text justification and formatting.

ROM routine entry point: $B39E

Routine operation: The position of the cursor on the line is obtained using routine $FFF0, which gets the value from location $D3. A zero is then put into the accumulator and the routine at $B391 used to put the value into FAC#1.

PRINT and PRINT# plus TAB(and SPC(

Abbreviated entry:

PRINT	?
PRINT#	P(shift)R
TAB(	T(shift)A
SPC(	S(shift)P

Tokens:

PRINT	Hex $99	Decimal 153
PRINT#	Hex $98	Decimal 152
TAB(	Hex $A3	Decimal 163
SPC(	Hex $A6	Decimal 166

Modes: Direct or program

Purpose: The PRINT and PRINT# commands will evaluate and then display on the current output device any string or numeric expression. The PRINT command will display to the screen and the PRINT# command to the currently opened output device. The output produced by PRINT and PRINT# can be formatted by use of the commands TAB(and SPC(plus either a comma or semicolon following the variable or literal.

Syntax: PRINT [arithmetic or string expression][arithmetic or string expression].....
PRINT# arithmetic expression, [arithmetic or string expression] [arithmetic or string expression].....

The first arithmetic expression following the PRINT# command is the logical file number of the designated output device, and must be a positive integer in the range 1 to 255. The following expression or expressions are the data to be output or displayed; if there is no expression then a carriage return is output. These expressions are identical in syntax for both PRINT and PRINT#; each of the expressions can be separated by the following formatting commands:

SPC(arithmetic expression) – moves the cursor position right by the number of characters indicated by the arithmetic expression
TAB(arithmetic expression) – moves the cursor to the character position number indicated by the evaluated expression
comma – a comma after a printed variable means that the following printed variable starts on the tenth column or a column divisible by 10
semicolon – this leaves the cursor at its current position thereby preventing a carriage return at the end of a line

With any of the above format commands, if the following output is a positive numeric value then a space is added in front of the number; all numeric values have a space added to the end of the value. It should be noted that the value following the SPC(and TAB(commands must evaluate to a positive integer in the range 0 to 255. It should also be noted that the TAB(and SPC(commands will not work after a PRINT#. The TAB(and SPC(commands both work by displaying the required number of cursor right characters. This means that these two commands will not delete any characters displayed in the area of screen over which the cursor jumps. Any variable or literal used in the PRINT command can include cursor, colour control and graphics characters.

Errors: String too long - if the length of the concatenated strings exceeds 255 characters
Device not present - no specified output device for PRINT#
Not output file - file not defined as output on PRINT#
Illegal quantity - number is outside the range on TAB and SPC

Use: This is the principal output command in Basic and has a very wide range of applications and uses. The additional formatting commands of standard Basic are fairly limited, and to overcome this the CTL command in Chapter 5 gives the programmer greater power over cursor positioning and general screen control.

ROM routine entry points: PRINT - $AAA0
PRINT# - $AA80
SPC(- $AAF8
TAB(- $AAF8

Routine operation: There are four different routes which can be taken by the PRINT routine and these depend on the character or command following the PRINT command. Interesting subroutines within the main PRINT routine are:

$AAA4 - test for TAB(branch if found
$AAA8 - test for SPC(branch if found
$AAAD - test for comma branch if found
$AAB1 - test for semicolon branch if found
$AABC - print numeral after converting to ASCII
$AAD7 - print CR or CRLF
$AA9D - print string
$AB1E - print string from memory at .a (lsb) and .y (msb)

It should be noted that the output device number is stored in location $13. On completion the buffer is reset and location $0200 is set to $00, .x is $FF and .y is $01.

The PRINT# command is just a simple subroutine call to $AA86 to perform the CMD operation and a jump to $ABB5, the end of the INPUT# routine. This restores the default I/O and sets location $13 to zero.

READ

Abbreviated entry: R(shift)E

Token: Hex $87 Decimal 135

Modes: Direct and program mode, but in direct mode a program must be present which contains DATA statements, otherwise an Out of data error will be generated.

Purpose: This command reads data stored in a DATA statement. Each time a READ command is executed it gets a different item from the list of data statements.

Syntax: READ variable [,variable][,variable]. Any valid variable type, both simple and array, can be assigned by the READ command. However, the variable type must match the data within the DATA statement otherwise a Type mismatch error will be created.

Errors: Out of data error - no more data statements within the program
Syntax - variable type does not match the data. This is flagged as being on the line containing the data and not on the line containing the READ. This kind of error would normally generate Type mismatch but there is a bug in the error routine of READ which generates the wrong error message and position.

Use: The READ command associated with data statements within a program is a very useful way of storing information and constants which are always required by the program. The only drawbacks to the DATA statement/READ method of data storage are firstly the difficulty of amending or adding further data whilst the program is running, and secondly that data elements are read serially. The first of these problems can be overcome using the DATA statement generator listed in Chapter 2 and the second limitation is overcome by the Restore to line routine in Chapter 4.

ROM routine entry point: $AC06

Routine operation: This routine is shared by both GET and INPUT. The three different functions are distinguished by the contents of $11. These values are:

GET - $40
INPUT - $00
READ - $98

These routines all scan the input buffer for blocks of data. In the case of GET a block of data is defined as a single character. For INPUT a terminating carriage return defines the input block. With READ the separating comma or end of line marker for the data statement pointed to by $41,$42 defines the data block. The block of data from whichever source is then assigned to the variable in the command. Of the entry points within this routine the following are interesting:

$AC0D	- INPUT entry point
$AC0F	- GET entry point
$AC71	- assign string to string variable
$AC89	- assign numeral to numeric variable
$ACB8	- used by READ to scan for DATA statements
$ACDF	- checks for terminating zero at end of buffer; if not found prints 'extra ignored' unless there is an active file, in which case no warning is given.

REM

Abbreviated entry: None

Token: Hex $8F Decimal 143

Modes: Direct and program

Purpose: This command allows comments to be added to a program; any text following the REM is ignored when the program is run but is listed on LIST.

Syntax: REM followed by any character

Errors: None

Use: Besides adding comments to Basic programs the REM command can be used for other purposes. One application is to store short blocks of data, which can be accessed by PEEK and POKE commands, or machine code subroutines in the text string following the REM command.

ROM routine entry point: $A93B

Routine operation: The routine to perform the REM command is part of the IF routine and is the same as that used for a condition 'false'. It skips the rest of the line by setting charget pointers $7A,$7B to the start of the next line by adding to their current contents the scan to the next line $A909 offset in the .y index register.

RESTORE

Abbreviated entry: RE(shift)S

Token: Hex $8C Decimal 140

Modes: Direct and program

Purpose: Resets the pointer to data statements in a Basic program to the first DATA statement.

Syntax: RESTORE has no following parameters

Errors: None

Use: The RESTORE command will reset the data statement pointer to the beginning of the program so that the READ command can start accessing data again from the beginning of the data statement table. The drawback of this is that RESTORE returns to the beginning of the data table; this means that if you wish to go back only a few items in the data table you must go back to the beginning and then use READ to scan back up again to the desired location. This restriction can be overcome by using the routine at the end of this section which performs a RESTORE to a given program line.

ROM routine entry point: $A81D

Routine operation: Sets the data statement pointer to the start of Basic program storage ($0800). This pointer is stored in locations $41,$42. This routine is also used by the RUN, CLR, and NEW routines.

RIGHT$

Abbreviated entry: R(shift)I

Token: Hex $C9 Decimal 201

Modes: Direct and program

Purpose: This takes the specified string and extracts from it a substring consisting of the specified number of characters at the right end of the string.

Syntax: RIGHT$(string expression,arithmetic expression). The string expression can be a string literal, string variable, a string function like LEFT$, or a combination of one or all of these, the only limitation being that the resulting string length must not exceed 255 characters. The arithmetic expression must be an integer number between 0 and 255 when evaluated.

Errors: Illegal quantity - value exceeds the limits 0 to 255

Use: The string functions are used extensively to manipulate strings and RIGHT$. The principal use is in getting rid of leading characters or truncating strings to a fixed length.

ROM routine entry point: $B72C

Routine operation: This pulls the parameter data off the stack and sets the string position pointer before jumping to the routine in LEFT$ at $B706, which creates the substring, stores it in memory and sets up the required pointers.

RND

Abbreviated entry: R(shift)N

Token: Hex $BB   Decimal 187

Modes: Direct and program

Purpose: This function generates a pseudo random number which it returns as a floating point fractional value in the range 0 to 1.

Syntax: RND(arithmetic expression). The expression is used as a seed for the random value calculation and can be any valid floating point number.

Errors: None

Use: The random numbers produced by the RND are not truly random. For a given seed value they will repeat the same sequence of values providing the random seed has not been reset with a RND(0). The seed value used in the RND function is important; a negative number will calculate a random number but will cause the next random number to have an identical value. A seed value of zero will set the seed to the contents of the timer in the CIA chip. This is the best way of generating a random value because it depends on the time since the machine was switched on and is thus unpredictable.

ROM routine entry point: $E097

Routine operation: A random value is created by this routine and stored in FAC#1. Prior to running this routine FAC#1 contains a 'seed' value used to initialise the random number calculation routine. The last random number generated is stored in locations $8B,$8F. If a zero argument is given in the RND function then the value in the CIA timers is used for the seed.

RUN

Abbreviated entry: R(shift)U

Token: Hex $8A   Decimal 138

Modes: Direct and program

Purpose: Initiates the execution of a Basic program either from the beginning of the program or from a specified line number.

Syntax: RUN [line number]. The line number is optional, but when specified it must be an existing line within the range of valid line numbers. If a line number follows RUN then program execution starts at the specified line number.

Errors: Undefined statement error – line specified after RUN does not exist
Syntax error – first byte of Basic program storage ($0800) or any end of line marker is not zero

Use: This initialises the execution of a Basic program. For a full explanation of how a program is executed see Chapter 1.

ROM routine entry point: $A871

Routine operation: If RUN is followed by a line number, then RUN calls the CLR routine to clear the contents of variables and stack, and jumps to the GOTO routine. If RUN is not followed by a line number then the charget pointers at $7A,$7B are set to the start of Basic program storage, the CLR routine is called, and the RUN initiated with a return to the main Basic control loop.

SAVE

Abbreviated entry: S(shift)A

Token: Hex $94 Decimal 148

Modes: Direct and program mode

Purpose: This command saves the contents of a specified section of memory onto an output device, either disk or tape.

Syntax: SAVE [string expression [,arithmetic expression [,arithmetic expression]]]. All the parameters within square brackets are optional; the string expression is the name of the program to be saved. When used with a disk drive the program name must always be used. The first arithmetic expression is the device number which is one for the tape drive and eight for disk etc., on the Commodore 64. The second arithmetic expression always follows the first and defines where the program will start in memory. If this value is zero, or no value is used, then the program will always be saved so that it will start loading at an address pointed to by the contents of the .x and .y index registers, normally the start of the Basic program storage area. If the second arithmetic expression is <>0 then the tape header will contain the address at which the program started. A secondary address of five will cause an end of tape block to be written; this has the effect of preventing the tape from reading past this block. The secondary address will have no effect on loading from tape if a secondary address of three is used in the SAVE command.

Errors: Device not present – specified device is not connected or device 0 or 3 designated
Missing file name – no file name was specified when loading from disk

Illegal device number - invalid device number
Illegal quantity - out of range device or secondary address values (range is 1 to 255)

Use: The functioning of this command depends whether it is used in direct or program mode. In direct mode the computer produces a series of messages which are displayed on the screen. These are:

Disk - SAVE "PROGRAM",8
SAVING "PROGRAM"
READY

Tape - SAVE ["PROGRAM"[,1[,0]]]
PRESS PLAY ON TAPE
SAVING "PROGRAM"
READY

On tape the square brackets denote that if the program name is not included in the SAVE command then it will not be recorded on the header or displayed in the messages. In program mode the only message displayed by the SAVE command is PRESS PLAY ON TAPE when saving to tape. The program will save correctly (see Program 13).

Source code for computed SAVE.

```
033C           !MEMORY SAVE ROUTINE
033C           !
C000           *=$C000
C000 20FDAE                   JSR $AEFD
C003 208AAD                   JSR $AD8A              !GET ADDRESS OF START
C006 20F7B7                   JSR $B7F7              ! INTO $14,$15
C009 A514                     LDA $14
C00B 85FB                     STA $FB
C00D A515                     LDA $15
C00F 85FC                     STA $FC
C011 20FDAE                   JSR $AEFD
C014 208AAD                   JSR $AD8A              !GET ADDRESS OF END
C017 20F7B7                   JSR $B7F7              !INTO $14,$15
C01A 20FDAE                   JSR $AEFD              !SCAN PAST COMMA
C01D 20D4E1                   JSR $E1D4              !GET FILE DETAILS
C020 A9FB                     LDA #$FB
C022 A614                     LDX $14
C024 A415                     LDY $15
C026 20D8FF                   JSR $FFD8              !SAVE FILE
C029 B001                     BCS ERROR
C02B 60                       RTS                    !DONE O.K.
C02C 4CF9E0 ERROR             JMP $E0F9
```

BASIC loader for computed SAVE.

```
10 INPUT"ADDRESS FOR MEMORY SAVE";I:S=I
20 READA:IFA=-1THEN50
30 POKEI,A:I=I+1
40 T=T+A:GOTO20
50 IFT<>6712THENPRINT"QQCHECKSUM ERROR :6712"T:END
60 IFI<>S+47THENPRINT"QQNUMBER OF DATA ERROR":END
70 PRINT"]QQUSE MEMORY SAVE TO SAVE BLOCKS OF MEMORY"
80 PRINT"QSYS("S"),START,END+1,"CHR$(34)"NAME"CHR$(34)"[,DEV]"
```

```
90 END
100 DATA32,253,174,32,138,173,32
110 DATA247,183,165,20,133,251,165
120 DATA21,133,252,32,253,174,32
130 DATA138,173,32,247,183,32,253
140 DATA174,32,212,225,169,251,166
150 DATA20,164,21,32,216,255,176
160 DATA1,96,76,249,224,-1
```

Program 13.

ROM routine entry point: $E156

Routine operation: This routine saves a program from the computer to disk or tape. The start address of the block of memory to be saved is stored in locations $2B,$2C (bottom of memory) and the end address of the SAVE is in locations $2D,$2E (start of variables). The file name and device number are obtained by the routine at $E1D4.

SGN

Abbreviated entry: S(shift)G

Token: Hex $B4 Decimal 180

Modes: Direct and program

Purpose: This function returns the sign of an arithmetic function; −1 if the expression is negative, 0 if zero, and +1 if positive.

Syntax: SGN(arithmetic expression). The expression must evaluate to a number within the permissible floating point value range.

Errors: Illegal quantity - value is out of range
Type mismatch - non numeric expression

Use: This command has fairly limited applications, mostly confined to performing conditional tests on values.

ROM routine entry point: $BC39

Routine operation: The routine to get the sign of FAC#1 is called ($BC2F). The sign of the value in FAC#1 is put into the msb of FAC#1, $88 is put into the exponent of FAC#1 and the rest of FAC#1 is zeroed.

SIN

Abbreviated entry: S(shift)I

Token: Hex $BF Decimal 191

Mode: Direct and program

Purpose: This command evaluates the sine of an angle in radians.

Syntax: SIN (arithmetic expression). The expression must be syntactically correct and within the range permissible for floating point numbers.

Errors: Syntax error – wrong command syntax e.g. missing closing bracket
Type mismatch – non arithmetic expression
Overflow error – expression is outside permissible floating point range

Use: This command is used within many trigonometric applications. It should be noted that the value of the expression must be in radians rather than degrees; an angle can be converted to radians by multiplying the angle by pi/180.

ROM routine entry point: $E26B

Routine operation: The argument in radians is stored in FAC#1. It is evaluated to give the sine of the angle, and this is stored in FAC#1.

SQR

Abbreviated entry: S(shift)Q

Token: Hex $BA Decimal 186

Mode: Direct and program

Purpose: Calculates the square root of a value.

Syntax: SQR(arithmetic expression). The arithmetic expression must be positive and within the normal range for floating point values.

Errors: Illegal quantity – value is negative

Use: This command is not essential since it can easily be replaced by the expression X ↑ .5, but the SQR function is convenient and slightly faster. When using machine code routines the SQR routine can easily be rewritten to use powers of any other value; this is because the routine uses a constant of .5 stored in memory as a floating point value. The pointers to this constant can easily be changed in a rewritten routine to point to a new constant (see *The Commodore 64 ROMs Revealed* for a listing of the routine).

ROM routine entry point: $BF71

Routine operation: The contents of FAC#1 (the argument) are transferred to FAC#2, FAC#1 is then loaded with the constant .5 (pointed to by .a and .y) and the routine jumps into the perform power routine at $BF78. The result is stored in FAC#1.

STOP

Abbreviated entry: S(shift)T

Token: Hex $90 Decimal 144

Mode: Direct and program

Purpose: Causes a program to exit from the program mode to the direct mode and print a message showing on which line the program stopped. This command is like END; typing CONT will allow the program to continue execution.

Syntax: STOP has no parameters but must always be followed by a colon or end of line marker.

Errors: Syntax error – if STOP is not followed by colon or end of line marker

Use: The STOP command can be used to set break points within the program during de-bugging, where a CONT will resume program execution.

ROM routine entry point: $A82F

Routine operation: This routine is shared with END (see END command for explanation).

STR$

Abbreviated entry: ST(shift)R

Token: Hex $C4 Decimal 196

Modes: Direct and program mode

Purpose: This command converts a number or numeric expression into a string.

Syntax: STR$(arithmetic expression). The arithmetic expression can evaluate to any floating point value within the permitted range. The resulting string will have the same format as that produced by PRINT when displaying the numeric variable.

Errors: Type mismatch – non numeric expression

Use: This command is used only to insert numeric values into strings, usually in association with a numeric formatting routine.

ROM routine entry point: $B465

Routine operation: The routine first checks that there is a numeric evaluation to the argument. The argument is stored in FAC#1, and this is converted into an ASCII string starting at location $0100 by the routine at $BDDF. The string and its related pointers are then set up in memory by the routine $B487.

SYS

Abbreviated entry: S(shift)Y

Token: Hex $9E Decimal 158

Modes: Direct and program

Purpose: This command transfers program control to a machine code program starting at the address following the SYS command. Control can be returned to Basic when an RTS gets the return address to the SYS routine off the stack.

Syntax: SYS arithmetic expression. The arithmetic expression must evaluate to a positive integer value within the range 0 to 65535; all non integers are rounded down.

Errors: Illegal quantity – address is outside the range 0 to 65535

Use: This is an essential command when calling machine code routines from a Basic program. The SYS command also allows the passing of parameters which will initialise the .x, .y, .a and status registers on entry to the machine code routine, and then save these same registers on exit. The contents of these registers are stored in the following memory locations:

$030C – save accumulator
$030D – save .x register
$030E – save .y register
$030F – save status register

ROM routine entry point: $E12A

Routine operation: This first gets a two byte value (the address) and puts it in locations $14 (lsb) $15 (msb), then pushes the return address to the stack followed by the processor status register from $030F, and loads the .a, .x, .y registers with the parameters stored in locations $030C to $030E. Control then jumps to the machine code routine using an indirect jump via locations $14,$15. On returning from the machine code routine the contents of the .a, .x, .y and status registers are saved in the above memory locations.

TAN

Abbreviated entry: None

Token: Hex $C0 Decimal 192

Modes: Direct and program

Purpose: This command evaluates the tangent of an angle in radians.

Syntax: TAN (arithmetic expression). The expression must be syntactically correct and within the range permissible for floating point numbers.

Errors: Syntax error - wrong command syntax e.g. missing closing bracket
Type mismatch - non arithmetic expression
Overflow error - expression is outside the permissible floating point range

Use: This command is used within many trigonometric applications. It should be noted that the value of the expression must be in radians rather than degrees; an angle can be converted to radians by multiplying the angle by pi/180.

ROM routine entry point: $E2B4

Routine operation: The argument in radians is stored in FAC#1. It is calculated by dividing the sine of the angle by the cosine, using the routines at $E26B (sine) and $E264 (cosine) to give the tangent of the angle; this is stored in FAC#1.

USR

Abbreviated entry: U(shift)S

Token: Hex $B7 Decimal 183

Modes: Direct and program

Purpose: This is an arithmetic function which will call a user written machine code routine.

Syntax: USR(arithmetic expression). The expression must evaluate to a value within the permissible range for floating point numbers.

Errors: Illegal quantity - if USR is not defined

Use: This command is useful when using machine code routines within a Basic program which involve passing parameters in full floating point form. The expression within the brackets following the USR command is evaluated and the result stored in FAC#1. This value can then be used by a machine code routine which starts as a jump routine to the actual routine. The jump is stored in three bytes from $0310 to $0312. If the jump is not set then it defaults on power up to give an illegal quantity error. On leaving the machine code routine the contents of FAC#1 are assigned to the variable on the other side of the equals sign.

ROM routine entry point: The routine will always jump to the vector jump starting at $0310.

Routine operation: As in all functions, the expression is first evaluated and the result stored in FAC#1. The routine then jumps to the vector jump in $0310 which has been set up by the programmer to point to the machine code subroutine. On encountering an RTS instruction terminating the machine code routine, the return address on stack transfers control to a routine where the contents of FAC#1 are assigned to the variable preceding the function.

VAL

Abbreviated entry: V(shift)A

Token: Hex $C5 Decimal 197

Modes: Direct and program

Purpose: This command converts a string or string expression into a numerical value; this command is the converse of STR$.

Syntax: VAL(string expression). The string command can consist of string variables, string literals, string functions like LEFT$, or a combination of these concatenated by a +. The maximum string length is one where the resulting number does not exceed the maximum permissible size of a floating point number. The resulting number will, if very large, be rounded and stored in exponent/mantissa form.

Errors: Overflow – resulting number exceeds the maximum range for floating point numbers
Type mismatch – non string expression

Use: This command is the converse of STR$ and is usually used in conjunction with this command. It should be noted that any spaces in the string are ignored, but if there is an alpha character in the string then all following numbers are ignored – unless that character is an E following a number, when the E is interpreted as indicating that the following number is an exponent.

ROM routine entry point: $B7AD

Routine operation: The string pointed to by charget pointers $7A,$7B is located and converted into a floating point number by the routine $BCF3; the result is stored in FAC#1.

VERIFY

Abbreviated entry: V(shift)E

Token: Hex $95 Decimal 149

Modes: Direct and program

Purpose: This command checks that the contents of a block of memory stored on tape or disk are identical to the current contents of the same block of memory. The VERIFY command is a special version of the LOAD command.

Syntax: The syntax is identical to LOAD.

Errors: Verify error – contents of the tape or disk do not match memory contents

Use: The VERIFY command is used principally to check that a program has been saved correctly. It does this by reading the program from tape or disk byte by byte and comparing it with the corresponding byte in memory. For this reason VERIFY cannot be used with data files, only with memory dumps. If the VERIFY is satisfactory then the computer gives an OK message, and when used in the program mode will continue executing the rest of the program.

ROM routine entry point: $E165

Routine operation: This routine sets the flag for 'verify' and continues with the LOAD routine. After the Kernal LOAD/VERIFY routine had been called, the status is checked to see if the VERIFY was correct. If so it prints OK, otherwise it gives an error message.

WAIT

Abbreviated entry: W(shift)A

Token: Hex $92 Decimal 146

Modes: Direct and program

Purpose: Halts the execution of a Basic program until the contents of a specified memory location have one or more bits set according to a bit pattern parameter.

Syntax: WAIT arithmetic expression, arithmetic expression [,arithmetic expression]. The first arithmetic expression is a memory location and must be a positive integer in the range 0 to 65535, the second arithmetic expression is the bit pattern to match and must therefore be a value in the range 0 to 255, the third optional parameter is another bit pattern matching byte which is ORed with the result of the second parameter and ANDed with the contents of memory; if the result is non zero then the WAIT loop is terminated.

Errors: Illegal quantity – first expression is outside the range 0 to 65535 and the second and third parameters are outside the range 0 to 255

Use: The format of the WAIT command is WAIT I,J,K. When this is executed, the contents of location I are ORed with K and ANDed with J. If the result of this is zero then the loop is repeated until it becomes non zero. The command is a test on bits in a memory location and the values in J and K would be powers of 2 (0,1,2,4,8,16,32,64,128,255 or a combination of these values). It should be noted that while the computer is in the WAIT loop the STOP key is not being tested and one should therefore be very careful that the bit combination chosen will occur. As an example of the use of WAIT the line:

```
100 GET A$: IF A$ = ““ THEN 100
```

can be replaced by:

```
100 WAIT 198,1:GETA$
```

This waits for a keypress before getting a character in A$. The WAIT command can also be used to test when the joystick is moved or when the fire button pressed. Another application is a timed pause using the timers in the CIA chip.

ROM routine entry point: $B82D

Routine operation: The two parameters are obtained using the routine at $B7EB. This leaves the address parameter in location $14,$15 and the second parameter in the .x index register. This second parameter is stored in $49, and the optional third parameter is then obtained by routine $B7F1 and stored in $4A; if there is no third parameter it defaults to zero. The routine then performs a loop which continues until the value at the location pointed to by $14,$15 is not equal to zero when exclusively ORed with the third parameter and ANDed with the second.

Chapter Four

BASIC Wedges and Vectors

This chapter covers the different types of wedge routine which can be used to intercept normal program execution and thereby be used to add extra commands to Basic or simply modify existing commands and operating system functions. All of the wedge programs, with the exception of the wedges into 'charget' and 'warm start' are required as the wedge routines for the extended Basic package in Chapter 5.

4.1 Charget

The charget routine is a short machine code routine located in zero page RAM memory which is used by the Basic interpreter to read the program, character by character, from memory. Charget occupies 24 bytes and starts at location $0073. The reason why charget is located in this part of RAM memory is that it contains a variable load address which is used to point to the current character to be accessed in the Basic program. This variable load address or pointer to source text is stored in locations $7A,$7B. There are two entry points to the charget routine. They are:

> **Charget** – entry point $0073. This gets the next character in the Basic program following the location pointed to by the address in $7A,$7B.
> **Chargot** – entry point $0079. This gets the character in the Basic program currently pointed to by the address in $7A,$7B.

The charget routine is designed to ignore spaces within a program, thus if the character accessed is a space, then the pointer in $7A,$7B is incremented and the following character accessed. (If that also is a space then this is continued until a non space character is reached.) The mode of the character is then checked before the character is passed to the calling routine in the accumulator. This mode check decides whether the character is numeric or not. If the character is numeric then the array flag in the processor status register is cleared, otherwise it is set. When using charget or any routines calling charget it is important to remember this use of the carry flag.

The charget routine is as follows:

```
Loc    Bytes                Operation          Comments

0073   E67A     CHARGET     INC $7A            ;increment the char-
                                               acter pointer lsb
0075   D002                 BNE CHARGOT        ;no rollover from lo byte
0077   E67B                 INC $7B            ;increment the high
                                               byte
0079   AD****   CHARGOT     LDA $****          ;get the byte into .A
007C   C93A                 CMP #$3A           ;is it colon?
007E   B00A                 BCS CHAREND        ;not numeric
0080   C920                 CMP #$20           ;is it a space?
0082   F0EF                 BEQ CHARGET        ;yes, ignore
0084   38                   SEC
0085   E930                 SBC #$30           ;set the carry for any
                                               value less than
0087   38                   SEC                ;#$30 which is the
                                               character for '0'
0088   E9D0                 SBC #$D0
008A   60       CHAREND     RTS                ;return to main routine
```

Note that the instruction at $0079 (CHARGOT) reads LDA $****. The **** indicate that the address in locations $7A,$7B is variable.

It is fairly easy to wedge into the charget subroutine, and such wedges are used in applications like a DOS wedge. Here a certain character, such as '@' is used to indicate that a wedge into current operation must occur, and the new routine executed. One good thing about wedging into charget is that any command can be trapped before it is executed. This is best done by replacing the first three bytes by JMP $zzzz, where zzzz is the address of the wedge routine. Then by pulling and pushing the two bytes of the return address one can find where charget was called from (example: HIMEM in Chapter 5). If charget was called by the execute statement routine, one can check that the next character is a wedge identifier character like '@'. If the next character is a wedge identifier then the required operation is performed, otherwise a JMP $0079 will return to chargot. It must be remembered that the charget pointer address in $7A,$7B must always be incremented before returning to the charget routine with a JMP $0079.

The chargot routine is best demonstrated by Program 14 which causes Basic to run in RAM and then modifies some of the command vectors to point to routines in the $C000 area of memory. Each of the modified routines uses a chargot at the beginning to check for a wedge identifier character(s). The routines that have been modified in this way are:

PRINT and **INPUT** (which now allow the positioning of the cursor by the @ character, thus simulating PRINT AT). Therefore to start printing a string A$ starting at the coordinates x,y on the screen, one could use the command:

```
PRINT @ x,y;A$
```

Source code for charget wedge.

```
C000             *=$C000
C000 A9A0                LDA #$A0                 !SET POINTERS
C002 85FC                STA $FC                  ! TO COPY BASIC
C004 A900                LDA #$00                 ! ROM INTO RAM
C006 85FB                STA $FB
C008 A000                LDY #$00
C00A B1FB   COPY1        LDA ($FB),Y              !GET BYTE
C00C 91FB                STA ($FB),Y              !STORE TO RAM
C00E C8                  INY
C00F D0F9                BNE COPY1                !UNTIL PAGE DONE
C011 E6FC                INC $FC                  !DO NEXT PAGE
C013 A5FC                LDA $FC
C015 C9C0                CMP #$C0                 !IF NEEDED
C017 D0F1                BNE COPY1
C019 A501                LDA $01                  !SWITCH OUT THE
C01B 29FE                AND #$FE                 ! BASIC ROM
C01D 8501                STA $01
C01F A246                LDX #$46                 !LOOP TO COPY
C021 BD2CC0 COPY2        LDA VECTOR-1,X           ! NEW VECTORS
C024 9D0BA0              STA $A00B,X              ! INTO PLACE
C027 CA                  DEX
C028 D0F7                BNE COPY2                !UNTIL DONE
C02A 4C74A4              JMP $A474                !BACK TO READY
C02D        !
C02D 30A841 VECTOR       BYT $30,$A8,$41,$A7,$1D,$AD,$F7,$A8
C035 A4AB                BYT $A4,$AB
C037 72C0                WOR INPUT-1
C039 80B005              BYT $80,$B0,$05,$AC,$A4,$A9,$9F,$A8
C041 70A827              BYT $70,$A8,$27,$A9
C045 C2C0                WOR RESTOR-1
C047 82A8D1              BYT $82,$A8,$D1,$A8,$3A,$A9,$2E,$A8
C04F 4AA92C              BYT $4A,$A9,$2C,$B8,$67,$E1,$55,$E1
C057 64E1B2              BYT $64,$E1,$B2,$B3
C05B 92C0                WOR POKE-1
C05D 7FAA                BYT $7F,$AA
C05F 82C0                WOR PRINT-1
C061 56A89B              BYT $56,$A8,$9B,$A6,$5D,$A6,$85,$AA
C069 29E1BD              BYT $29,$E1,$BD,$E1,$C6,$E1,$7A,$AB
C071 41A6                BYT $41,$A6
C073        !
C073 207900 INPUT        JSR $0079                !CURRENT CHAR
C076 C940                CMP #$40                 !IS IT '@'?
C078 F003                BEQ INPUT1               !YES
C07A 4CBFAB              JMP $ABBF                !DO INPUT
C07D 20F2C0 INPUT1       JSR POSIT                !POSITION CURSOR
C080 4CBFAB              JMP $ABBF                !DO INPUT
C083        !
C083 207900 PRINT        JSR $0079                !CURRENT CHAR
C086 C940                CMP #$40                 !IS IT '@'?
C088 F003                BEQ PRINT1               !YES
C08A 4CA0AA              JMP $AAA0                !DO PRINT
C08D 20F2C0 PRINT1       JSR POSIT                !POSITION CURSOR
C090 4CA0AA              JMP $AAA0                !DO PRINT
C093        !
C093 207900 POKE         JSR $0079                !CURRENT CHAR
C096 C921                CMP #$21                 !IS IT '!'?
C098 F003                BEQ POKE1                !YES
C09A 4C24B8              JMP $B824                !DO POKE
C09D 207300 POKE1        JSR $0073                !NEXT CHAR
C0A0 208AAD              JSR $AD8A                !GET ADDRESS
C0A3 20F7B7              JSR $B7F7                !FIX IT
C0A6 A514                LDA $14
C0A8 85FB                STA $FB
C0AA A515                LDA $15
C0AC 85FC                STA $FC
C0AE 20FDAE              JSR $AEFD                !SCAN ','
```

```
C0B1 208AAD                JSR $AD8A               !GET VALUE
C0B4 20F7B7                JSR $B7F7               !FIX IT
C0B7 A000                  LDY #$00
C0B9 A514                  LDA $14                 !GET LO BYTE
C0BB 91FB                  STA ($FB),Y             !STORE IT
C0BD C8                    INY
C0BE A515                  LDA $15                 !GET HI BYTE
C0C0 91FB                  STA ($FB),Y             !STORE IT
C0C2 60                    RTS                     !DONE
C0C3          !
C0C3 207900 RESTOR         JSR $0079               !CURRENT CHAR
C0C6 9003                  BCC REST1               !NUMERIC
C0C8 4C1DA8                JMP $A81D               !DO RESTORE
C0CB D003   REST1          BNE REST2
C0CD 4C1DA8                JMP $A81D               !DO RESTORE
C0D0 208AAD REST2          JSR $AD8A               !GET LINE NUMBER
C0D3 20F7B7                JSR $B7F7               !FIX IT
C0D6 2013A6                JSR $A613               !FIND BASIC LINE
C0D9 A55F                  LDA $5F                 !GET LO BYTE
C0DB D002                  BNE REST3
C0DD C660                  DEC $60                 !DECREMENT HI BYTE
C0DF C65F   REST3          DEC $5F                 !DECREMENT LO BYTE
C0E1 A55F                  LDA $5F                 !ADDRESS LO
C0E3 8541                  STA $41
C0E5 A560                  LDA $60                 !ADDRESS HI
C0E7 8542                  STA $42
C0E9 A514                  LDA $14                 !LINE # LO
C0EB 853F                  STA $3F
C0ED A515                  LDA $15                 !LINE # HI
C0EF 8540                  STA $40
C0F1 60                    RTS
C0F2          !
C0F2 207300 POSIT          JSR $0073               !NEXT CHAR
C0F5 209EB7                JSR $B79E               !GET X POSITION
C0F8 86FB                  STX $FB
C0FA 20F1B7                JSR $B7F1               !GET Y POSITION
C0FD E019                  CPX #25                 !0-24?
C0FF 9003                  BCC POSIT1
C101 4C48B2 POSERR         JMP $B248               !'ILLEGAL QUANTITY'
C104 A4FB   POSIT1         LDY $FB                 !GET X POS
C106 C028                  CPY #40                 !0-39?
C108 B0F7                  BCS POSERR
C10A 18                    CLC
C10B 20F0FF                JSR $FFF0               !POSITION CURSOR
C10E 4C7900                JMP $0079               !GET CHAR & EXIT POSIT
```

BASIC loader for charget wedge.

```
10 I=49152
20 READA:IFA=-1THEN50
30 POKEI,A:I=I+1
40 T=T+A:GOTO20
50 IFT<>36249THENPRINT"ⓆⓆⓆCHECKSUM ERROR - 36249"T:END
60 PRINT"ⓆⓆⓆYOU NOW HAVE 4 MORE COMMANDS:"
70 PRINT"ⓆPRINT @X,Y......":PRINT"ⓆINPUT @X,Y......"
80 PRINT"ⓆPOKE !AD,2VAL":PRINT"ⓆRESTORE LINNUM"
90 SYS49152
100 DATA169,160,133,252,169,0,133
110 DATA251,160,0,177,251,145,251
120 DATA200,208,249,230,252,165,252
130 DATA201,192,208,241,165,1,41
140 DATA254,133,1,162,70,189,44
150 DATA192,157,11,160,202,208,247
160 DATA76,116,164,48,168,65,167
170 DATA29,173,247,168,164,171,114
180 DATA192,128,176,5,172,164,169
190 DATA159,168,112,168,39,169,194
200 DATA192.130,168,209,168,58,169
```

```
210 DATA46,168,74,169,44,184,103
220 DATA225,85,225,100,225,178,179
230 DATA146,192,127,170,130,192,86
240 DATA168,155,166,93,166,133,170
250 DATA41,225,189,225,198,225,122
260 DATA171,65,166,32,121,0,201
270 DATA64,240,3,76,191,171,32
280 DATA242,192,76,191,171,32,121
290 DATA0,201,64,240,3,76,160
300 DATA170,32,242,192,76,160,170
310 DATA32,121,0,201,33,240,3
320 DATA76,36,184,32,115,0,32
330 DATA138,173,32,247,183,165,20
340 DATA133,251,165,21,133,252,32
350 DATA253,174,32,138,173,32,247
360 DATA183,160,0,165,20,145,251
370 DATA200,165,21,145,251,96,32
380 DATA121,0,144,3,76,29,168
390 DATA208,3,76,29,168,32,138
400 DATA173,32,247,183,32,19,166
410 DATA165,95,208,2,198,96,198
420 DATA95,165,95,133,65,165,96
430 DATA133,66,165,20,133,63,165
440 DATA21,133,64,96,32,115,0
450 DATA32,158,183,134,251,32,241
460 DATA183,224,25,144,3,76,72
470 DATA178,164,251,192,40,176,247
480 DATA24,32,240,255,76,121,0
490 DATA-1
```

Program 14.

POKE has a check for '!' enabling a two byte poke (see DOKE in Chapter 5). To use this command to put a value into two consecutive memory locations use the following command syntax:

POKE ! address, two byte value

RESTORE is the other command to be changed. This checks for any character that is not a colon or end of line. If so, then a line number is read in to bring about a restore to line number. This command has the following syntax:

RESTORE line number

An example of using this command is shown in Program 15.

```
10 READ A,B,C
20 RESTORE110
30 READ D,E
40 PRINTA;B;C;D;E
50 END
100 DATA 1
110 DATA 2
120 DATA 3
```

Program 15.

4.2 Warm start vector wedge

The warm start routine is a loop routine which waits for the entry of a program line or direct command. When not actually running a program the computer will always be in this warm start loop. The Basic warm start vector ($0300) contains the entry address of the warm start routine; this is used as an indirect jump to the warm start by the other interpreter routines. Since this indirect or vector jump address is stored in RAM it can be changed to point to another routine. One example of this is to use the warm start vector to protect a Basic program from being listed or otherwise accessed outside a normal run mode. When the program is running the Basic warm start vector is changed to point to $FCE2 and any program break-in will cause the computer to cold start. This will reset all system variables to power up values and NEW the program.

Program 16 will save a Basic program so that it will automatically run, when loaded using a short machine code routine, and be protected from unauthorised break-in. It should be noted that following a LOAD the computer will return to the warm start loop. The routine utilises this and the warm start vector change.

Source code for warm start vector wedge.

```
C000          *=$C000
C000 A52B                LDA $2B                  !GET START OF BASIC
C002 8D17C1              STA STBAS                ! AND STORE AWAY
C005 A52C                LDA $2C
C007 8D18C1              STA STBAS+1
C00A A9A5                LDA #$A5                 !SET START OF
C00C 852B                STA $2B                  ! AUTO RUN CODE
C00E 8D0203              STA $0302                ! AND WARM START
C011 A902                LDA #$02                 ! ENTRY POINT
C013 852C                STA $2C
C015 8D0303              STA $0303
C018 A52D                LDA $2D                  !GET END OF BASIC
C01A 8D19C1              STA ENDBAS               ! AND STORE AWAY
C01D A52E                LDA $2E
C01F 8D1AC1              STA ENDBAS+1
C022 A903                LDA #$03                 !SET END OF
C024 852E                STA $2E                  ! AUTO RUN CODE
C026 A904                LDA #$04
C028 852D                STA $2D
C02A A256                LDX #$56                 !LOOP VALUE FOR CODE
C02C BDC0C0 AUTO1        LDA AUTOCD,X             !GET AUTO BYTE
C02F 9DA502              STA $02A5,X              ! AND STORE IT
C032 CA                  DEX
C033 10F7                BPL AUTO1                !AND NEXT BYTE
C035 A908                LDA #$08                 !POINTER TO PROGRAM
C037 85FC                STA $FC
C039 A900                LDA #$00
C03B 85FB                STA $FB
C03D A001                LDY #$01
C03F B1FB   AUTO2        LDA ($FB),Y              !GET BYTE
C041 49FF                EOR #$FF                 !NOT IT
C043 91FB                STA ($FB),Y              !STORE IT
C045 C8                  INY
C046 D0F7                BNE AUTO2                !UNTIL END OF PAGE
C048 E6FC                INC $FC                  !DO NEXT PAGE
C04A A5FC                LDA $FC                  ! UNTIL END OF BASIC
C04C C9A0                CMP #$A0                 ! STORAGE
C04E D0EF                BNE AUTO2
C050 20D4E1              JSR $E1D4                !GET FILE NAME
C053 A903                LDA #$03                 !SET SEC ADDRS
C055 85B9                STA $B9                  ! FOR LOAD
C057 2059E1              JSR $E159                !SAVE AUTO RUN CODE
C05A          !
```

```
C05A AD17C1          LDA STBAS       !RESTORE BASIC POINTERS
C05D 852B            STA $2B
C05F AD18C1          LDA STBAS+1
C062 852C            STA $2C
C064 AD19C1          LDA ENDBAS
C067 852D            STA $2D
C069 AD1AC1          LDA ENDBAS+1
C06C 852E            STA $2E
C06E A987            LDA #<SAVE      !SET SAVE VECTOR
C070 8D3203          STA $0332
C073 A9C0            LDA #>SAVE
C075 8D3303          STA $0333
C078 A983            LDA #$83        !RESET WARM START
C07A 8D0203          STA $0302
C07D A9A4            LDA #$A4
C07F 8D0303          STA $0303
C082 A900            LDA #$00        !SET RUN MODE
C084 859D            STA $9D
C086 60              RTS
C087          !
C087 A9ED     SAVE   LDA #$ED        !RESET SAVE VECTOR
C089 8D3203          STA $0332
C08C A9F5            LDA #$F5
C08E 8D3303          STA $0333
C091 A901            LDA #$01        !DEVICE TAPE
C093 AA              TAX
C094 A8              TAY
C095 20BAFF          JSR $FFBA
C098 A901            LDA #$01        !LENGTH OF NAME
C09A A2A6            LDX #$A6        !POINTER TO NAME
C09C A002            LDY #$02
C09E 20BDFF          JSR $FFBD
C0A1 2059E1          JSR $E159       !SAVE FILE
C0A4 A900            LDA #$00
C0A6 85FB            STA $FB
C0A8 A908            LDA #$08
C0AA 85FC            STA $FC
C0AC A001            LDY #$01
C0AE B1FB     SAVE2  LDA ($FB),Y     !DECODE PROGRAM
C0B0 49FF            EOR #$FF
C0B2 91FB            STA ($FB),Y
C0B4 C8              INY
C0B5 D0F7            BNE SAVE2
C0B7 E6FC            INC $FC
C0B9 A5FC            LDA $FC
C0BB C9A0            CMP #$A0
C0BD D0EF            BNE SAVE2
C0BF 60              RTS
C0C0          !
C0C0 A983     AUTOCD LDA #$83        !RESET WARM START
C0C2 8D0203          STA $0302
C0C5 A9A4            LDA #$A4
C0C7 8D0303          STA $0303
C0CA A900            LDA #$00        !SET RUN MODE
C0CC 859D            STA $9D
C0CE 20D5FF          JSR $FFD5       !DUMMY LOAD
C0D1 A901            LDA #$01
C0D3 AA              TAX
C0D4 A8              TAY
C0D5 20BAFF          JSR $FFBA       !SET FILE DETAILS
C0D8 A901            LDA #$01
C0DA A2A6            LDX #$A6
C0DC A002            LDY #$02
C0DE 20BDFF          JSR $FFBD       !SET NAME DETAILS
C0E1 A900            LDA #$00
C0E3 20D5FF          JSR $FFD5       !LOAD
C0E6 862D            STX $2D         !SET VARIABLE POINTERS
```

```
C0E8 862F                    STX $2F
C0EA 8631                    STX $31
C0EC 842E                    STY $2E
C0EE 8430                    STY $30
C0F0 8432                    STY $32
C0F2 A000                    LDY #$00
C0F4 84FB                    STY $FB
C0F6 A908                    LDA #$08
C0F8 85FC                    STA $FC
C0FA C8                      INY
C0FB A9FF    AUTOC1          LDA #$FF                    !DECODE PROGRAM
C0FD 51FB                    EOR ($FB),Y
C0FF 91FB                    STA ($FB),Y
C101 C8                      INY
C102 D0F7                    BNE AUTOC1
C104 E6FC                    INC $FC
C106 A5FC                    LDA $FC
C108 C9A0                    CMP #$A0
C10A D0EF                    BNE AUTOC1
C10C A900                    LDA #$00
C10E 205EA6                  JSR $A65E                   !SET CHARGET POINTERS
C111 208EA6                  JSR $A68E                   !PERFORM 'CLR'
C114 4CAEA7                  JMP $A7AE                   !EXECUTE STATEMENT
C117 0000    STBAS           WOR 0
C119 0000    ENDBAS          WOR 0
```

BASIC loader for warm start vector wedge.

```
1000 I=49152:T=0
1010 READA:IFA=-1THEN1040
1020 POKEI,A:T=T+A
1030 I=I+1:GOTO1010
1040 IFT<>37131THENPRINT"CHECKSUM ERROR "37131,T:END
1050 IFI<>49431THENPRINT"NUMBER OF DATA VALUE ERROR":END
1060 PRINT"[illegible]TO SAVE A PROGRAM WITH AUTO RUN,"
1070 PRINT"[illegible]LOAD THE PROGRAM AND ENTER:"
1080 PRINT"[illegible]SYS(49152)"CHR$(34)"FILENAME"CHR$(34)":SAVE":END
1090 DATA165,43,141,23,193,165,44
1100 DATA141,24,193,169,165,133,43
1110 DATA141,2,3,169,2,133,44
1120 DATA141,3,3,165,45,141,25
1130 DATA193,165,46,141,26,193,169
1140 DATA3,133,46,169,4,133,45
1150 DATA162,86,189,192,192,157,165
1160 DATA2,202,16,247,169,8,133
1170 DATA252,169,0,133,251,160,1
1180 DATA177,251,73,255,145,251,200
1190 DATA208,247,230,252,165,252,201
1200 DATA160,208,239,32,212,225,169
1210 DATA3,133,185,32,89,225,173
1220 DATA23,193,133,43,173,24,193
1230 DATA133,44,173,25,193,133,45
1240 DATA173,26,193,133,46,169,135
1250 DATA141,50,3,169,192,141,51
1260 DATA3,169,131,141,2,3,169
1270 DATA164,141,3,3,169,0,133
1280 DATA157,96,169,237,141,50,3
1290 DATA169,245,141,51,3,169,1
1300 DATA170,168,32,186,255,169,1
1310 DATA162,121,160,192,32,189,255
1320 DATA32,89,225,169,0,133,251
1330 DATA169,8,133,252,160,1,177
1340 DATA251,73,255,145,251,200,208
1350 DATA247,230,252,165,252,201,160
1360 DATA208,239,96,169,131,141,2
1370 DATA3,169,164,141,3,3,169
1380 DATA0,133,157,32,213,255,169
```

```
1390 DATA1,170,168,32,186,255,169
1400 DATA1,162,166,160,2,32,189
1410 DATA255,169,0,32,213,255,134
1420 DATA45,134,47,134,49,132,46
1430 DATA132,48,132,50,160,0,132
1440 DATA251,169,8,133,252,200,169
1450 DATA255,81,251,145,251,200,208
1460 DATA247,230,252,165,252,201,160
1470 DATA208,239,169,0,32,94,166
1480 DATA32,142,166,76,174,167,-1
```

Program 16.

The following routines are the start of the Basic extension commands. These are the main control routines that patch the extra commands into the Commodore 64's Basic. They should be used in the order in which they appear.

Initialisation

This file contains the initialisation routines and the table of added commands and their vectors. The commands are initialised by calling the cold start ($FCE2 – 64738) which is a simulation of power-up. The routines cannot be used with a cartridge as they take up the same memory locations and simulate a cartridge.

The routine labelled 'COLD' is the actual power-up routine and the routine labelled 'WRST' is the NMI routine that makes sure that the function keys and lister are not disabled.

```
LOC   CODE        LINE


0000                      .LIB INITRT
0000                      * =$8000
8000  7A 80               .WOR COLD       ;COLD START ENTRY
8002  39 80               .WOR WRST       ;RESTORE ENTRY
8004  C3                  .BYT $C3,$C2,$CD,'80'
8005  C2
8006  CD
8007  38 30
8009              ;
8009  8B E3       LINK    .WOR $E38B
800B  83 A4               .WOR $A483
800D  C9 81               .WOR CRNCHT
800F  9E 82               .WOR PRINT
8011  F7 82               .WOR HANDLE
8013  34 83               .WOR ARITH
8015              ;
8015  4C 48 B2    VECTOR JMP $B248         ;USR JUMP
8018  00                  .BYT 0
8019  31 EA               .WOR $EA31      ;IRQ
801B  44 80               .WOR WRST01     ;BREAK
801D  47 FE               .WOR $FE47      ;NMI
801F  4A F3               .WOR $F34A      ;OPEN
8021  91 F2               .WOR $F291      ;CLOSE
8023  0E F2               .WOR $F20E      ;SET INPUT
8025  50 F2               .WOR $F250      ;SET OUTPUT
8027  33 F3               .WOR $F333      ;RESTORE I/O
8029  E3 83               .WOR LISTER     ;INPUT
802B  CA F1               .WOR $F1CA      ;OUTPUT
802D  ED F6               .WOR $F6ED      ;TEST-STOP
```

```
LOC    CODE          LINE

802F   3E F1                .WOR $F13E      ;GET
8031   2F F3                .WOR $F32F      ;ABORT I/O
8033   44 80                .WOR WRST01     ;WARM RESTART
8035   A5 F4                .WOR $F4A5      ;LOAD
8037   ED F5                .WOR $F5ED      ;SAVE
8039                 ;
8039   20 BC F6      WRST   JSR $F6BC        ;UPDATE TIME
803C   20 E1 FF             JSR $FFE1       ;STOP KEY?
803F   F0 03                BEQ WRST01      ;YES
8041   4C 72 FE             JMP $FE72       ;NO
8044   20 A3 FD      WRST01 JSR $FDA3        ;INIT I/O
8047   20 18 E5             JSR $E518       ;INIT VIC CHIP
804A   20 5D 80             JSR SETKER      ;INIT KERNAL VECTORS
804D   20 CC FF             JSR $FFCC       ;RESTORE I/O
8050   A9 00                LDA #$00
8052   85 13                STA $13         ;INPUT PROMPT FLAG
8054   20 7A A6             JSR $A67A       ;INIT BASIC
8057   58                   CLI             ;ENABLE IRQ
8058   A2 80         WRST02 LDX #$80         ;SET FOR READY
805A   4C 88 E3             JMP $E388       ;GO TO READY
805D                 ;
805D   A2 15         SETKER LDX #<VECTOR     ;POINT TO
805F   A0 80                LDY #>VECTOR    ;KERNAL VECTORS
8061   86 C3                STX $C3
8063   84 C4                STY $C4
8065   A0 23                LDY #$23        ;LOOP TO COPY VECTORS
8067   B1 C3         STKER1 LDA ($C3),Y      ;GET BYTE
8069   99 10 03             STA $0310,Y     ;STORE IT
806C   88                   DEY
806D   10 F8                BPL STKER1      ;AND NEXT
806F   A9 68                LDA #<FUNC      ;POINT TO FUNCTION
8071   A0 83                LDY #>FUNC      ;KEY ROUTINE
8073   8D 8F 02             STA $028F       ;STORE IN KEYBOARD
8076   8C 90 02             STY $0290       ;TABLE SETUP VECTOR
8079   60                   RTS
807A                 ;
807A   8E 16 D0      COLD   STX $D016        ;SHRINK SCREEN
807D   20 A3 FD             JSR $FDA3       ;INIT I/O
8080   20 50 FD             JSR $FD50       ;INIT SYSTEM CONSTANTS
8083   20 5B FF             JSR $FF5B
8086   20 5D 80             JSR SETKER      ;SET KERNAL VECTORS
8089   58                   CLI             ;ENABLE IRQ
808A   20 E5 80             JSR SETBAS      ;SET BASIC VECTORS
808D   20 BF E3             JSR $E3BF       ;INIT BASIC
8090   A9 80                LDA #$80        ;SET TOP OF RAM
8092   85 34                STA $34
8094   85 36                STA $36
8096   85 38                STA $38
8098   A9 00                LDA #$00
809A   85 33                STA $33
809C   85 35                STA $35
809E   85 37                STA $37
80A0   A9 AC                LDA #<POWER     ;POINT TO POWER
80A2   A0 80                LDY #>POWER     ;UP MESSAGE
80A4   20 2D E4             JSR $E42D       ;OUTPUT MESSAGE
80A7   A2 FB                LDX #$FB
80A9   9A                   TXS             ;SET STACK POINTER
80AA   D0 AC                BNE WRST02      ;ALWAYS
80AC                 ;
80AC   93            POWER  .BYT $93,$0D
80AD   0D
80AE   20 20                .BYT '    **** EXTENDED 64 BASIC'
80C8   20 56                .BYT ' V01 ****',$0D,$0D
80D1   0D
80D2   0D
80D3   20 36                .BYT ' 64K RAM SYSTEM  ',$00
```

```
LOC   CODE       LINE


80E4  00
80E5                 ;
80E5  A2 0B          SETBAS LDX #$0B         ;LOOP
80E7  BD 09 80       STBAS1 LDA LINK,X       ;GET BYTE
80EA  9D 00 03              STA $0300,X     ;STORE IT
80ED  CA                    DEX
80EE  10 F7                 BPL STBAS1      ;DO NEXT
80F0  60                    RTS
80F1                 ;
80F1  52 55          CLIST  .BYT 'RU',$CE
80F3  CE
80F4  43 54                 .BYT 'CT',$CC
80F6  CC
80F7  41 50                 .BYT 'APPEN',$C4
80FC  C4
80FD  41 55 54              .BYT 'AUT',$CF
8100  CF
8101  43 41                 .BYT 'CATALO',$C7
8107  C7
8108  43 48                 .BYT 'CHANG',$C5
810D  C5
810E  43 48                 .BYT 'CHAI',$CE
8112  CE
8113  43 52                 .BYT 'CRUNC',$C8
8118  C8
8119  44 45                 .BYT 'DELET',$C5
811E  C5
811F  44 49 53              .BYT 'DIS',$CB
8122  CB
8123  44 4F 4B              .BYT 'DOK',$C5
8126  C5
8127  44 55 4D              .BYT 'DUM',$D0
812A  D0
812B  45 58 45              .BYT 'EXE',$C3
812E  C3
812F  46 49 4E              .BYT 'FIN',$C4
8132  C4
8133  47 45                 .BYT 'GE',$D4
8135  D4
8136  4B 45                 .BYT 'KE',$D9
8138  D9
8139  4D 41                 .BYT 'MA',$D4
813B  D4
813C  4D 45                 .BYT 'MERG',$C5
8140  C5
8141  4F 4C                 .BYT 'OL',$C4
8143  C4
8144  50 4F                 .BYT 'PO',$D0
8146  D0
8147  50 55                 .BYT 'PU',$D4
8149  D4
814A  52 45                 .BYT 'RENUMBE',$D2
8151  D2
8152  52 45                 .BYT 'REPEA',$D4
8157  D4
8158  53 4F 52              .BYT 'SOR',$D4
815B  D4
815C  54 52                 .BYT 'TRACEO',$CE
8162  CE
8163  54 52                 .BYT 'TRACEOF',$C6
816A  C6
816B  54 59 50              .BYT 'TYP',$C5
816E  C5
816F  55 4E                 .BYT 'UNTI',$CC
8173  CC
```

```
LOC   CODE          LINE


8174                 ;
8174  44 45 45                .BYT 'DEE',$CB
8177  CB
8178  48 49                   .BYT 'HIME',$CD
817C  CD
817D  4C 4F                   .BYT 'LOME',$CD
8181  CD
8182  56 41                   .BYT 'VARPT',$D2
8187  D2
8188  00                      .BYT 0
8189                 ;
8189  18 9D          CADDR    .WOR RUN-1
818B  AA 88                   .WOR CTL-1
818D  D7 84                   .WOR APPEND-1
818F  36 85                   .WOR AUTONO-1
8191  B5 85                   .WOR CATLOG-1
8193  B7 86                   .WOR CHANGE-1
8195  83 86                   .WOR CHAIN-1
8197  FE 87                   .WOR CRUNCH-1
8199  AC 89                   .WOR DELETE-1
819B  4C 8A                   .WOR DISK-1
819D  DE 8A                   .WOR DOKE-1
819F  01 8B                   .WOR DUMP-1
81A1  CD 8C                   .WOR EXEC-1
81A3  92 8D                   .WOR FIND-1
81A5  D0 8E                   .WOR GET-1
81A7  13 90                   .WOR KEY-1
81A9  17 91                   .WOR MAT-1
81AB  AF 97                   .WOR MERGE-1
81AD  84 98                   .WOR OLD-1
81AF  BA 98                   .WOR POP-1
81B1  79 99                   .WOR PUT-1
81B3  5C 9A                   .WOR RENUMB-1
81B5  F1 9C                   .WOR REPEAT-1
81B7  24 9D                   .WOR SORT-1
81B9  FC 9E                   .WOR TRON-1
81BB  42 9F                   .WOR TROFF-1
81BD  4F 9F                   .WOR TYPE-1
81BF  6B 9F                   .WOR UNTIL-1
81C1                 ;
81C1  94 89                   .WOR DEEK-1
81C3  DE 8F                   .WOR HIMEM-1
81C5  5C 90                   .WOR LOMEM-1
81C7  C9 9F                   .WOR VARPTR-1
81C9                 ;
81C9                 FNSTRT   =29
81C9                          .END
```

Crunch to tokens

This routine is wedged into the crunch token link at locations $0304–$0305 (772–773). Crunch to tokens will take the input line and convert all command words to one (for normal Basic) or two (for extended Basic) byte token values. This does exactly the same as the original Basic version except that the extended keyword table is checked before the normal Basic table.

Crunch to tokens is performed directly after the warm start routine encounters a carriage return, no matter whether the command is in direct mode or for entering or deleting a line in memory.

```
LOC     CODE         LINE


81C9                        .LIB CRUNCH-TOKEN
81C9                 ; CRUNCH KEYWORD LINK
81C9                 ;      FOR USE WITH THE ROUTINES IN
81C9                 ; 'ADVANCED COMMODORE 64 BASIC REVEALED'
81C9                 ;
81C9    A6 7A        CRNCHT LDX $7A
81CB    A0 04               LDY #$04
81CD    84 0F               STY $0F
81CF    BD 00 02     CRNC01 LDA $0200,X      ;GET CHAR
81D2    10 07               BPL CRNC02      ;CHAR IS OK
81D4    C9 FF               CMP #$FF        ;PIPRINT
81D6    F0 2B               BEQ CRNC08      ;YES,SEND IT
81D8    E8                  INX             ;NO, ILLEGAL CHAR
81D9    D0 F4               BNE CRNC01      ; SO DO NEXT
81DB                 ;
81DB    C9 20        CRNC02 CMP #$20         ;SPACEPRINT
81DD    F0 24               BEQ CRNC08      ;YES, SEND IT
81DF    85 08               STA $08
81E1    C9 22               CMP #$22        ;QUOTESPRINT
81E3    F0 47               BEQ CRNC12      ;YES, SCAN QUOTE END
81E5    24 0F               BIT $0F
81E7    70 1A               BVS CRNC08      ;SEND CHAR
81E9    C9 3F               CMP #$3F        ;'PRINT' PRINT
81EB    D0 04               BNE CRNC03      ;NO
81ED    A9 99               LDA #$99        ;SET TO PRINT TOKEN
81EF    D0 12               BNE CRNC08      ;SEND IT
81F1                 ;
81F1    C9 30        CRNC03 CMP #$30         ;<0 PRINT
81F3    90 04               BCC CRNC04      ;YES, HUNT FOR KEYWORD
81F5    C9 3C               CMP #$3C        ;< '<' PRINT
81F7    90 0A               BCC CRNC08      ;YES, SEND CHAR
81F9    4C 46 82     CRNC04 JMP CRNC15       ;HUNT FOR KEYWORD
81FC                 ;
81FC    A9 EE        CRNC05 LDA #$EE         ;ONE OF MINE
81FE    2C                  .BYT $2C        ;SKIP NEXT 2 BYTES
81FF                 ;
81FF    05 0B        CRNC06 ORA $0B          ;ONE OF BASIC'S
8201    A4 71        CRNC07 LDY $71          ;RESTORE Y
8203    E8           CRNC08 INX              ;NEXT POSITION
8204    C8                  INY
8205    99 FB 01            STA $01FB,Y     ;STORE IT
8208    C9 EE               CMP #$EE        ;MINEPRINT
820A    F0 31               BEQ CRNC14      ;YES, SEND 2ND BYTE
820C    B9 FB 01            LDA $01FB,Y     ;NO, END OF INPUTPRINT
820F    F0 22               BEQ CRNC13      ;YES
8211    38                  SEC
8212    E9 3A               SBC #$3A        ;':' PRINT
8214    F0 04               BEQ CRNC09      ;YES
8216    C9 49               CMP #$49        ;DATA ?
8218    D0 02               BNE CRNC10      ;NO
821A    85 0F        CRNC09 STA $0F
821C    38           CRNC10 SEC
821D    E9 55               SBC #$55        ;REM ?
821F    D0 AE               BNE CRNC01      ;NO DO NEXT CHAR
8221    85 08               STA $08         ;SET QUOTE FLAG
8223    BD 00 02     CRNC11 LDA $0200,X      ;GET BYTE
8226    F0 DB               BEQ CRNC08      ;END OF INPUT, SEND
8228    C5 08               CMP $08         ;QUOTE FLAGPRINT
822A    F0 D7               BEQ CRNC08      ;YES, SEND
822C    C8           CRNC12 INY              ;STORE CHAR
822D    99 FB 01            STA $01FB,Y
8230    E8                  INX
8231    D0 F0               BNE CRNC11      ;DO NEXT
8233                 ;
8233    99 FD 01     CRNC13 STA $01FD,Y      ;STORE ZERO
```

```
LOC   CODE         LINE


8236  C6 7B               DEC $7B
8238  A9 FF               LDA #$FF
823A  85 7A               STA $7A
823C  60                  RTS            ;EXIT CRUNCH
823D               ;
823D  A5 0B        CRNC14 LDA $0B         ;GET 2ND BYTE
823F  C8                  INY
8240  99 FB 01            STA $01FB,Y    ;STORE IT
8243  4C CF 81            JMP CRNC01     ;DO NEXT BYTE
8246               ;
8246  84 71        CRNC15 STY $71         ;SAVE OFF Y
8248  A0 FF               LDY #$FF
824A  86 7A               STX $7A        ; AND X POINTERS
824C  CA                  DEX
824D  A9 01               LDA #$01       ;START TOKEN VAL=1
824F  85 0B               STA $0B
8251  C8           CRNC16 INY
8252  E8                  INX
8253  BD 00 02     CRNC17 LDA $0200,X     ;GET BYTE
8256  38                  SEC
8257  F9 F1 80            SBC CLIST,Y    ;AS KEYWORD TABLEPRINT
825A  F0 F5               BEQ CRNC16     ;YES, CHECK NEXT
825C  C9 80               CMP #$80       ;SHIFT OUTPRINT
825E  F0 9C               BEQ CRNC05     ;YES, FOUND
8260  A6 7A               LDX $7A        ;RESTORE BUFFER POINTER
8262  E6 0B               INC $0B        ;NEXT TOKEN
8264  C8           CRNC18 INY
8265  B9 F0 80            LDA CLIST-1,Y  ;END OF KEYWORDPRINT
8268  10 FA               BPL CRNC18     ;NO
826A  B9 F1 80            LDA CLIST,Y    ;END OF TABLEPRINT
826D  D0 E4               BNE CRNC17     ;NO, CHECK NEXT
826F  A0 00               LDY #$00       ;START TOKEN AT 0
8271  84 0B               STY $0B        ;FOR BASIC
8273  88                  DEY
8274  A6 7A               LDX $7A        ;GET INPUT POINTER
8276  CA                  DEX
8277  C8           CRNC19 INY
8278  E8                  INX
8279  BD 00 02     CRNC20 LDA $0200,X     ;GET BYTE
827C  38                  SEC
827D  F9 9E A0            SBC $A09E,Y    ;AS IN TABLEPRINT
8280  F0 F5               BEQ CRNC19     ;YES, CHECK NEXT
8282  C9 80               CMP #$80       ;SHIFT OUTPRINT
8284  D0 03               BNE CRNC21     ;NO, TRY NEXT WORD
8286  4C FF 81            JMP CRNC06     ;YES, SEND BASIC TOKEN
8289  A6 7A        CRNC21 LDX $7A         ;RESTORE INPUT POINTER
828B  E6 0B               INC $0B        ;NEXT TOKEN
828D  C8           CRNC22 INY
828E  B9 9D A0            LDA $A09D,Y    ;END OF WORDPRINT
8291  10 FA               BPL CRNC22     ;NO
8293  B9 9E A0            LDA $A09E,Y    ;END OF TABLEPRINT
8296  D0 E1               BNE CRNC20     ;NO, TRY NEXT WORD
8298  BD 00 02            LDA $0200,X    ;ELSE SEND BYTE
829B  4C 01 82            JMP CRNC07
829E                      .END
```

Tokens to text

This routine is wedged into the print token link at locations $0306-$0307 (774-775). Tokens to text is used in the list command only to convert any token value (greater than 127 for normal Basic or preceded by $EE-238 for extended Basic) back into the command word and print it to the output device.

```
LOC   CODE       LINE


829E                    .LIB PRINT-TOKEN
829E             ; PRINT TOKENS LINK
829E             ;     FOR USE WITH THE ROUTINES IN
829E             ; 'ADVANCED COMMODORE 64 BASIC REVEALED'
829E             ;
829E  30 03      PRINT  BMI PRIN02        ;A TOKEN
82A0  4C F3 A6   PRIN01 JMP $A6F3         ;PRINT IT
82A3  C9 FF      PRIN02 CMP #$FF          ;IS IT PI?
82A5  F0 F9             BEQ PRIN01       ;YES
82A7  24 0F             BIT $0F          ;QUOTES?
82A9  30 F5             BMI PRIN01       ;YES
82AB  C9 EE             CMP #$EE         ;ONE OF MINE?
82AD  F0 05             BEQ PRIN08       ;DO MINE
82AF  20 D9 82          JSR PRIN09       ;DO BASIC
82B2  30 03             BMI PRIN13       ;ALWAYS
82B4  20 BA 82   PRIN08 JSR PRIN03        ;DO MINE
82B7  4C EF A6   PRIN13 JMP $A6EF         ;AND NEXT
82BA             ;
82BA  C8         PRIN03 INY               ;GET TOKEN CHAR
82BB  B1 5F             LDA ($5F),Y
82BD  AA                TAX
82BE  84 49             STY $49          ;SAVE Y
82C0  A0 FF             LDY #$FF
82C2  CA         PRIN04 DEX
82C3  F0 08             BEQ PRIN06       ;FOUND IT
82C5  C8         PRIN05 INY
82C6  B9 F1 80          LDA CLIST,Y      ;GET CHAR FROM TABLE
82C9  10 FA             BPL PRIN05       ;UNTIL END OF WORD
82CB  30 F5             BMI PRIN04       ;FOUND END OF WORD
82CD  C8         PRIN06 INY
82CE  B9 F1 80          LDA CLIST,Y      ;GET CHAR FROM TABLE
82D1  30 05             BMI PRIN07       ;LAST CHAR OF WORD
82D3  20 D2 FF          JSR $FFD2        ;PRINT IT
82D6  D0 F5             BNE PRIN06       ;NEXT CHAR
82D8  60         PRIN07 RTS               ;DO LAST
82D9             ;
82D9  38         PRIN09 SEC
82DA  E9 7F             SBC #$7F         ;REMOVE SHIFT
82DC  AA                TAX
82DD  84 49             STY $49          ;SAVE .Y
82DF  A0 FF             LDY #$FF
82E1  CA         PRIN10 DEX
82E2  F0 08             BEQ PRIN12       ;FOUND IT
82E4  C8         PRIN11 INY
82E5  B9 9E A0          LDA $A09E,Y      ;GET CHAR FROM TABLE
82E8  10 FA             BPL PRIN11       ;UNTIL END OF WORD
82EA  30 F5             BMI PRIN10       ;FOUND END OF WORD
82EC  C8         PRIN12 INY
82ED  B9 9E A0          LDA $A09E,Y      ;GET CHAR FROM TABLE
82F0  30 E6             BMI PRIN07       ;LAST CHAR OF WORD
82F2  20 D2 FF          JSR $FFD2        ;PRINT CHAR
82F5  D0 F5             BNE PRIN12       ;ALWAYS
82F7                    .END
```

Execute statement

This routine is wedged into the start new Basic code link at locations $0308-$0309 (776-777). This is the control part of the main Basic interpreter loop. It takes a token value and executes the routine via the vector table in the initialisation file. There is a special case routine for PRINT which uses the same token as in normal Basic but the routine has been rewritten to allow the CTL command.

```
LOC   CODE        LINE


82F7                     .LIB HANDLE-TOKEN
82F7              ; EXECUTE STATEMENT LINK
82F7              ;      FOR USE WITH THE ROUTINES IN
82F7              ; 'ADVANCED COMMODORE 64 BASIC REVEALED'
82F7              ;
82F7  20 73 00    HANDLE JSR $0073       ;GET CODE
82FA  C9 EE              CMP #$EE       ;IS IT MY TOKEN?
82FC  F0 0A              BEQ HAND01     ;YES, DO IT
82FE  C9 99              CMP #$99       ;IS IT PRINT?
8300  F0 1F              BEQ DOPRNT     ;YES
8302  20 79 00           JSR $0079      ;GET CURRENT CHAR
8305  4C E7 A7           JMP $A7E7      ;DO BASIC CODE
8308              ;
8308  20 0E 83    HAND01 JSR HAND02      ;EXECUTE THE CODE
830B  4C AE A7           JMP $A7AE      ;AND NEXT
830E              ;
830E  20 8B 8E    HAND02 JSR FIND13      ;GET TOKEN CHAR
8311  38                 SEC
8312  E9 01              SBC #$01
8314  0A                 ASL A          ;TIMES 2
8315  A8                 TAY
8316  B9 8A 81           LDA CADDR+1,Y  ;GET HI BYTE
8319  48                 PHA            ;TO STACK
831A  B9 89 81           LDA CADDR,Y    ;GET LO BYTE
831D  48                 PHA            ;TO STACK
831E  4C 73 00           JMP $0073      ;EXECUTE IT
8321              ;
8321              ;PRINT SPECIAL CASE
8321              ;
8321  20 27 83    DOPRNT JSR HAND03      ;DO PRINT COMMAND
8324  4C AE A7           JMP $A7AE      ;DO NEXT COMMAND
8327              ;
8327  AD 33 83    HAND03 LDA PADDR+1     ;GET HI BYTE
832A  48                 PHA            ;TO STACK
832B  AD 32 83           LDA PADDR      ;GET LO BYTE
832E  48                 PHA            ;TO STACK
832F  4C 73 00           JMP $0073      ;EXECUTE PRINT
8332  DA 98       PADDR  .WOR PRINTT-1   ;VECTOR FOR PRINT
8334                     .END
```

Execute arithmetic

This routine is wedged into the arithmetic link at locations $030A–030B (778–779). This routine is called by the evaluate expression and transfers control to one of the four arithmetic routines included in this package. If the extended Basic command is not one of the four arithmetic routines, a Syntax error is output.

```
LOC   CODE        LINE


8334                     .LIB ARITH-TOKEN
8334              ; ARITHMETIC LINK
8334              ;      FOR USE WITH THE ROUTINES IN
8334              ; 'ADVANCED COMMODORE 64 BASIC REVEALED'
8334              ;
8334  A9 00       ARITH  LDA #$00        ;TYPE FLAG TO NUMERIC
8336  85 0D              STA $0D
8338  20 73 00           JSR $0073      ;GET BYTE
```

```
LOC   CODE        LINE


833B  C9 EE              CMP #$EE        ;ONE OF MINE?
833D  F0 06              BEQ ARITH1      ;YES
833F  20 79 00           JSR $0079       ;GET CURRENT CHAR
8342  4C 8D AE           JMP $AE8D       ;OPERATE
8345                ;
8345  20 8B 8E      ARITH1 JSR FIND13     ;GET TOKEN CHAR
8348  C9 1D              CMP #FNSTRT     ;IS IT A FUNCTION
834A  B0 03              BCS ARITH2      ;YES
834C  4C 08 AF           JMP $AF08       ;SYNTAX ERROR
834F                ;
834F  85 24         ARITH2 STA $24        ;SAVE TOKEN VAL
8351  A9 AD              LDA #$AD        ;SETUP RETURN ADDRESS
8353  48                 PHA
8354  A9 8C              LDA #$8C
8356  48                 PHA
8357  C6 24              DEC $24
8359  A5 24              LDA $24         ;GET TOKEN
835B  0A                 ASL A           ;TIMES 2
835C  AA                 TAX
835D  BD 8A 81           LDA CADDR+1,X   ;GET HI BYTE
8360  48                 PHA
8361  BD 89 81           LDA CADDR,X     ;GET LO BYTE
8364  48                 PHA
8365  4C 73 00           JMP $0073       ;EXECUTE FUNCTION
8368                     .END
```

Function keys

This routine is wedged into the keyboard table set-up vector at locations $028F–$0290 (655–656). The routine checks whether the computer is in direct or program mode; if in direct mode then the normal routine is executed, if in program mode the quotes flag is checked, and if set the normal routine is executed.

The current key pressed is checked for one of the four function keys and the shift key. If it was a function key the text for that key is read from behind the Basic ROM, and put into the keyboard buffer until all eight characters or a zero byte terminator are found. If it was not a function key the normal routine is executed.

```
LOC   CODE        LINE


8368                     .LIB FUNC-KEYS
8368  A5 9D       FUNC   LDA $9D          ;DIRECT?
836A  F0 10              BEQ FUNC01      ;NO
836C  A9 01              LDA #$01        ;QUOTES?
836E  24 D4              BIT $D4
8370  D0 0A              BNE FUNC01      ;YES, IGNORE
8372  A5 CB              LDA $CB         ;KEY PRESSED
8374  C9 03              CMP #$03        ;F7?
8376  90 04              BCC FUNC01      ;NO, LESS THAN
8378  C9 07              CMP #$07        ;F5?
837A  90 03              BCC FUNC02      ;YES, IS A FUNCTION KEY
837C  4C 48 EB    FUNC01 JMP $EB48        ;DO NORMAL KEYS
837F              ;
837F  C5 C5       FUNC02 CMP $C5          ;ALREADY DONE?
8381  F0 F9              BEQ FUNC01      ;YES
8383  A9 00              LDA #$00        ;CLEAR POINTER
```

```
LOC    CODE          LINE


8385   85 FC                STA $FC
8387   85 FB                STA $FB
8389   A9 01                LDA #$01        ;SHIFT KEY?
838B   2C 8D 02             BIT $028D
838E   F0 04                BEQ FUNC03      ;NO
8390   A9 20                LDA #$20
8392   85 FB                STA $FB
8394   A9 BF         FUNC03 LDA #$BF         ;ADD START OF STORE
8396   85 FC                STA $FC         ; TO POINTER
8398   A9 C0                LDA #$C0
839A   18                   CLC
839B   65 FB                ADC $FB
839D   85 FB                STA $FB
839F   A5 CB                LDA $CB
83A1   C9 03                CMP #$03        ;F7?
83A3   D0 04                BNE FUNC04      ;NO
83A5   A9 18                LDA #24
83A7   D0 12                BNE FUNC07
83A9   C9 06         FUNC04 CMP #$06         ;F5?
83AB   D0 04                BNE FUNC05      ;NO
83AD   A9 10                LDA #16
83AF   D0 0A                BNE FUNC07
83B1   C9 05         FUNC05 CMP #$05         ;F3?
83B3   D0 04                BNE FUNC06      ;NO
83B5   A9 08                LDA #8
83B7   D0 02                BNE FUNC07
83B9   A9 00         FUNC06 LDA #$00         ;MUST BE F1
83BB   18            FUNC07 CLC              ;SET VAL INTO POINTER
83BC   65 FB                ADC $FB
83BE   85 FB                STA $FB
83C0   A0 00                LDY #$00
83C2   A9 36                LDA #$36        ;SWITCH OUT BAS ROM
83C4   85 01                STA $01
83C6   B1 FB         FUNC08 LDA ($FB),Y      ;GET CHAR
83C8   F0 08                BEQ FUNC09      ;ZERO BYTE TERMINATOR
83CA   99 77 02             STA $0277,Y     ;STORE IN BUFFER
83CD   C8                   INY
83CE   C0 08                CPY #$08        ;ALL 8?
83D0   D0 F4                BNE FUNC08      ;NOT YET
83D2   84 C6         FUNC09 STY $C6          ;#CHARS IN BUFFER
83D4   A9 37                LDA #$37        ;PUT BASIC ROM BACK
83D6   85 01                STA $01
83D8   A5 CB                LDA $CB         ;SET LAST=PRESENT
83DA   85 C5                STA $C5         ; KEYS,
83DC   AD 8D 02             LDA $028D       ; SHIFT COMBO
83DF   8D 8E 02             STA $028E
83E2   60                   RTS             ;ALL DONE
83E3                        .END
```

Program lister

This routine is wedged into the INPUT vector at locations $0324–0325 (804–805). This routine completely simulates the normal input routine. First the input device is checked for keyboard input and if it is not so the normal routine is executed. Direct mode is then checked for and if it is not, the normal routine is executed.

The next part of the routine is copied directly from the kernal routine except that the cursor down key is checked for and, if found, the cursor position is checked. If the cursor is not on the bottom line of the screen, the cursor down character is printed. If the cursor is on the bottom line, instead of printing cursor down the next line number is found and that line is listed (for any output device). (*Note*: There is no check for quotes, which means that if you are entering a line on the bottom line of the screen, the line will be wiped out and a line listed if you press the cursor down key even from within quotes.)

When the last line of the program is listed the cursor will remain at the end of the line. Pressing the cursor down key again will produce the message:

************ END OF PROGRAM ************

After this, the program will start listing from the beginning again.

```
LOC    CODE        LINE


83E3                       .LIB LISTER
83E3   A5 99       LISTER  LDA $99
83E5   D0 04               BNE LIST01       ;NOT KEYBOARD
83E7   A5 9D               LDA $9D
83E9   D0 03               BNE LIST02       ;IS DIRECT INPUT
83EB   4C 57 F1    LIST01  JMP $F157         ;DO NORMAL
83EE                ;
83EE   A5 D3       LIST02  LDA $D3           ;SAVE CURRENT CURSOR
83F0   85 CA               STA $CA          ; COLUMN
83F2   A5 D6               LDA $D6
83F4   85 C9               STA $C9          ; AND ROW
83F6   98                  TYA              ;SAVE .X AND .Y
83F7   48                  PHA
83F8   8A                  TXA
83F9   48                  PHA
83FA   A5 D0               LDA $D0          ;SCREEN OR KEYBOARD?
83FC   F0 06               BEQ LIST04       ;KEYBOARD
83FE   4C 3A E6            JMP $E63A        ;DO FOR SCREEN
8401                ;
8401   20 16 E7    LIST03  JSR $E716         ;DISPLAY CHAR TO SCREEN
8404   A5 C6       LIST04  LDA $C6           ;ANY CHARS IN BUFFER?
8406   85 CC               STA $CC          ;IF NOT, BLINK CURSOR
8408   8D 92 02            STA $0292        ;AUTO SCROLL DOWN
840B   F0 F7               BEQ LIST04       ;REPEAT UNTIL CHAR
840D   78                  SEI              ;DISABLE KEYBOARD
840E   A5 CF               LDA $CF          ;CURSOR BLINK?
8410   F0 0C               BEQ LIST05       ;NO
8412   A5 CE               LDA $CE          ;RESTORE ORIGINAL CHAR
8414   AE 87 02            LDX $0287        ; AND COLOUR
8417   A0 00               LDY #$00
8419   84 CF               STY $CF          ;SWITCH OFF BLINK
841B   20 13 EA            JSR $EA13        ;RESTORE
841E   20 B4 E5    LIST05  JSR $E5B4         ;REMOVE CHAR FROM BUFFER
8421   C9 83               CMP #$83         ;RUN/STOP?
8423   D0 10               BNE LIST07       ;NO
8425   A2 09               LDX #$09         ;COPY TEXT INTO BUFFER
8427   78                  SEI
8428   86 C6               STX $C6
842A   BD E6 EC    LIST06  LDA $ECE6,X
842D   9D 76 02            STA $0276,X
8430   CA                  DEX
8431   D0 F7               BNE LIST06       ;REPEAT UNTIL ALL DONE
8433   F0 CF               BEQ LIST04       ;DONE, OPERATE ON RUN/STOP
8435   C9 0D       LIST07  CMP #$0D          ;CARRIAGE RETURN?
8437   D0 03               BNE LIST08       ;NO
8439   4C 02 E6            JMP $E602        ;END OF INPUT
843C                ;
843C   C9 11       LIST08  CMP #$11          ;CURSOR DOWN?
843E   D0 C1               BNE LIST03       ;NO GET NEXT CHAR
```

```
LOC   CODE        LINE


8440  A6 D6               LDX $D6
8442  E0 18               CPX #24       ;SCROLL SCREEN?
8444  F0 03               BEQ LIST09    ;YES
8446  4C 01 84            JMP LIST03    ;NO, NEXT CHAR
8449  A2 18       LIST09  LDX #24        ;SET CURSOR TO
844B  A0 00               LDY #$00      ; BEGINNING OF LINE
844D  18                  CLC
844E  20 F0 FF            JSR $FFF0
8451  E6 14               INC $14       ;FIND NEXT LINE TO
8453  D0 02               BNE LIST10    ; LIST
8455  E6 15               INC $14+1
8457  20 13 A6    LIST10  JSR $A613      ;GET ADDRESS
845A  A0 01               LDY #$01
845C  B1 5F               LDA ($5F),Y   ;END OF PROGRAM?
845E  D0 10               BNE LIST11    ;NO
8460  A9 FF               LDA #$FF
8462  85 14               STA $14       ;NEXT LINE NUMBER=0
8464  85 15               STA $14+1
8466  A9 AB               LDA #<EOFMES  ;TELL USER THAT THE
8468  A0 84               LDY #>EOFMES  ; END OF PROGRAM HAS
846A  20 1E AB            JSR $AB1E     ; BEEN REACHED
846D  4C 04 84            JMP LIST04    ;GET NEXT CHAR
8470  A0 02       LIST11  LDY #$02       ;GET LINE NUMBER
8472  B1 5F               LDA ($5F),Y   ;LO BYTE
8474  85 14               STA $14
8476  C8                  INY
8477  B1 5F               LDA ($5F),Y   ;HI BYTE
8479  85 15               STA $14+1
847B  A9 94               LDA #<LIST12  ;RETURN TO LIST12
847D  8D 00 03            STA $0300     ;AFTER LIST
8480  A9 84               LDA #>LIST12
8482  8D 01 03            STA $0301
8485  68                  PLA           ;SAVE 2 BYTES IN
8486  8D A9 84            STA STACK     ; SAFE LOCATION
8489  68                  PLA
848A  8D AA 84            STA STACK+1
848D  A0 01               LDY #$01
848F  84 0F               STY $0F
8491  4C D7 A6            JMP $A6D7     ;LIST LINE
8494  A9 8B       LIST12  LDA #$8B       ;RESET ERROR VECTOR
8496  8D 00 03            STA $0300
8499  A9 E3               LDA #$E3
849B  8D 01 03            STA $0301
849E  AD AA 84            LDA STACK+1   ;RESTORE 2 BYTES
84A1  48                  PHA
84A2  AD A9 84            LDA STACK
84A5  48                  PHA
84A6  4C 04 84            JMP LIST04    ;DO NEXT CHAR
84A9  00 00       STACK   .WOR 0
84AB  0D          EOFMES  .BYT $0D,$0D,$12
84AC  0D
84AD  12
84AE  2A 2A               .BYT '************ END OF PROGRAM ************'
84D6  0D                  .BYT $0D,$00
84D7  00
84D8                      .END
```

Chapter Five

Extended BASIC - A Complete Package

Introduction

This chapter contains a collection of programs which will create 31 extra commands to the Commodore 64's Basic and modify two other commands. These extra commands will be of considerable use to any Basic programmer. The commands require the wedge programs in Chapter 4 to be loaded as part of the assembly; these wedges allow the following commands to be used as ordinary Basic commands. The commands and a description of their use is given in the documentation accompanying each of the routines. All these extra commands and their associated wedge, tokenising the parsing routines are designed to be stored in the cartridge ROM area of $8000 up for an area of just under 8K of memory. The routines are designed to emulate a ROM cartridge based program and will thus power up on cold start. The listings are all in CBM assembler format. For readers wishing to obtain these programs in machine readable form, they are available as both source and object code at an inclusive cost of £10 from: Advanced Commodore 64 BASIC Revealed Software Offer, 40 Bowling Green Lane, London EC1. (Please make cheques payable to Zifra Software Ltd.)

The extended Basic commands are:

APPEND
CHANGE
DELETE
EXEC
KEY
OLD
RENUMBER
TYPE

AUTO
CRUNCH
DISK
FIND
LOMEM
POP
REPEAT & RUN
UNTIL

CATALOG
CTL
DOKE
GET
MAT
PRINT
SORT
VARPTR

CHAIN
DEEK
DUMP
HIMEM
MERGE
PUT
TRACE ON &
TRACE OFF

APPEND

Abbreviated entry: A(shift)P

Affected Basic abbreviations: None

Token: Hex $EE,$03   Decimal 238,3

Modes: Direct and program

Recommended mode: Direct

Purpose: To load a program into memory so that it appears 'on top' of the current program. This routine will work with both disk and cassette and the variable pointers when loaded are set to the end of the combined program. When this routine is used, you should check that the line numbers of the APPENDed program are larger than the line numbers of the program in memory.

Syntax: APPEND [filename[,d[,s]]] – where d is the device number and s is the secondary address.

Errors: The same errors will be encountered as in the Basic command LOAD.

Use: This routine would be used mostly to add Basic library routines onto the end of your programs. It would be used rather than MERGE because of the difference in speed. APPEND is much faster than MERGE.

Routine entry point: $84D8

Routine operation: The APPEND routine uses LOAD's parameter parsing routine to get the filename etc., then sets the secondary address so that it loads at the end of the Basic program in memory. The load routine is then called, the program is re-chained and variable pointers are set.

```
LOC   CODE         LINE


84D8                      .LIB APPEND
84D8  A9 00        APPEND LDA #$00
84DA  85 0A               STA $0A
84DC  20 D4 E1            JSR $E1D4        ; GET FILE PARAMETERS
84DF  A9 00               LDA #$00
84E1  85 B9               STA $B9          ;SET SA FOR ALT LOAD
84E3  A5 2D               LDA $2D
84E5  38                  SEC
84E6  E9 02               SBC #$02         ; SET LOAD ADDRESS
84E8  AA                  TAX              ; DIRECTLY AFTER RESIDENT
84E9  A5 2E               LDA $2D+1        ; PROGRAM.
84EB  E9 00               SBC #$00
84ED  A8                  TAY
84EE  A5 0A               LDA $0A
84F0  20 D5 FF            JSR $FFD5        ; LOAD
84F3                ;
84F3  20 33 A5     RESVAR JSR $A533         ;RE-CHAIN LINES
84F6  A5 2D               LDA $2D
84F8  A4 2E               LDY $2D+1        ; RESET VARIABLE
84FA  38                  SEC              ; POINTERS TO END OF
84FB  E9 02               SBC #$02         ; NEW PROGRAM
84FD  85 57               STA $57
84FF  98                  TYA
8500  E9 00               SBC #$00
8502  85 58               STA $57+1
8504  A0 00        RESV01 LDY #$00          ; FIND END OF PROGRAM
8506  B1 57               LDA ($57),Y      ; AND SET VARIABLE
8508  D0 1B               BNE RESV02       ; POINTERS
850A  C8                  INY
850B  B1 57               LDA ($57),Y
```

```
LOC   CODE        LINE


850D  D0 16             BNE RESV02
850F  A5 57             LDA $57
8511  18                CLC
8512  69 02             ADC #$02
8514  85 2D             STA $2D
8516  85 2F             STA $2F
8518  85 31             STA $31
851A  A5 58             LDA $57+1
851C  69 00             ADC #$00
851E  85 2E             STA $2D+1
8520  85 30             STA $2F+1
8522  85 32             STA $31+1
8524  60                RTS
8525  A0 00       RESV02 LDY #$00        ; NOT YET END OF
8527  B1 57             LDA ($57),Y     ; PROGRAM. GET
8529  85 59             STA $59         ; ADDRESS OF NEXT
852B  C8                INY             ; LINE.
852C  B1 57             LDA ($57),Y
852E  85 58             STA $57+1
8530  A5 59             LDA $59
8532  85 57             STA $57
8534  4C 04 85          JMP RESV01
8537                    .END
```

AUTO

Abbreviated entry: A(shift)U

Affected Basic abbreviations: None

Token: Hex $EE,$04   Decimal 238,4

Modes: Direct and program

Recommended mode: Direct only

Purpose: To save time when entering a program by providing the user with the next line number to be entered. To enable the AUTO line numbering, enter AUTO followed by the line number increment. To disable AUTO just enter AUTO without a number. The next line number is picked up from the previous line typed in, so if you enter a line 10 with the auto step at 10, the next line number will be 20. If you change this number to, say, 100 and enter that line, the next line number displayed will be 110. A new line number is not displayed if nothing is entered on the line.

Syntax: AUTO [step]

Errors: Syntax error – if the step value is greater than 63999
(maximum line number)

Use: The command is used in direct mode to enable or disable AUTO line numbering. When enabled, AUTO will produce line numbers after entering a line until it is disabled with AUTO without an increment value. If you wish to exit from the AUTO facility when a line number has been displayed, either press return (which will delete that line if it exists), or cursor down off that line.

Routine entry point: $8537

Routine operation: First this routine checks to see if there is a number following it. If not it will disable AUTO, otherwise it will read the number and store as the step and enable AUTO. The actual routine is wedged into the crunch tokens link. It first checks that the first non space character in the input buffer is a numeric character and sets a flag to say yes or no. The line is then tokenised and if there was no line number or there was nothing following the line number, the routine exits. If the previous line typed in had a line number with something following it, the line number is read from the pointer. The step is then added to it, and the number converted to ASCII and inserted into the keyboard buffer.

```
LOC     CODE          LINE


8537                          .LIB AUTO
8537    F0 18         AUTONO  BEQ AUTOFF       ; NO STEP, TURN OFF
8539    20 6B A9              JSR $A96B        ; GET STEP
853C    A5 14                 LDA $14          ; STORE AWAY
853E    8D 5C 85              STA AUTOST
8541    A5 15                 LDA $15
8543    8D 5D 85              STA AUTOST+1
8546    A9 5E                 LDA #<AUTO       ; ENABLE AUTO
8548    8D 04 03              STA $0304
854B    A9 85                 LDA #>AUTO
854D    8D 05 03              STA $0305
8550    60                    RTS
8551                  ;
8551    A9 C9         AUTOFF  LDA #<CRNCHT      ; DISABLE AUTO
8553    8D 04 03              STA $0304
8556    A9 81                 LDA #>CRNCHT
8558    8D 05 03              STA $0305
855B    60                    RTS
855C    00 00         AUTOST  .WOR 0
855E                  ;
855E    AD 00 02      AUTO    LDA $0200         ; CHECK FIRST CHARACTER
8561    C9 30                 CMP #$30         ; IN INPUT BUFFER FOR
8563    90 0A                 BCC AUTO01       ; A NUMBER
8565    C9 3A                 CMP #$3A
8567    B0 06                 BCS AUTO01
8569    A9 01                 LDA #$01         ; SET FLAG TO SAY
856B    85 02                 STA $02          ; DO IT
856D    D0 04                 BNE AUTO02
856F    A9 00         AUTO01  LDA #$00          ; SET FLAG TO SAY
8571    85 02                 STA $02          ; DON'T DO IT
8573    20 C9 81      AUTO02  JSR CRNCHT        ; CRUNCH INPUT
8576    A5 02                 LDA $02          ; CHECK FLAG
8578    D0 01                 BNE AUTO03
857A    60                    RTS              ; DON'T DO IT
857B    C0 05         AUTO03  CPY #$05          ; CHECK FOR BLANK
857D    D0 01                 BNE AUTO04       ; INPUT LINE
857F    60                    RTS
8580    AD 5C 85      AUTO04  LDA AUTOST        ; ADD STEP TO PREVIOUS
8583    18                    CLC              ; LINE NUMBER
8584    65 14                 ADC $14
8586    AA                    TAX
8587    AD 5D 85              LDA AUTOST+1
858A    65 15                 ADC $15
858C    86 63                 STX $63
```

```
LOC   CODE          LINE

858E  85 62                STA $62
8590  A2 90                LDX #$90
8592  38                   SEC
8593  98                   TYA
8594  48                   PHA
8595  20 49 BC             JSR $BC49      ; CONVERT LINE NUMBER
8598  20 DF BD             JSR $BDDF      ; TO ASCII STRING
859B  85 FB                STA $FB
859D  84 FC                STY $FC
859F  A0 00                LDY #$00
85A1  B1 FB         AUTO05 LDA ($FB),Y     ; COPY ASCII
85A3  F0 06                BEQ AUTO06     ; STRING INTO KYBD
85A5  99 77 02             STA $0277,Y    ; BUFFER
85A8  C8                   INY
85A9  D0 F6                BNE AUTO05
85AB  C8            AUTO06 INY
85AC  A9 20                LDA #$20       ; AND A SPACE
85AE  99 77 02             STA $0277,Y
85B1  84 C6                STY $C6        ; NUMBER OF CHARS IN
85B3  68                   PLA            ; BUFFER
85B4  A8                   TAY
85B5  60                   RTS
85B6                       .END
```

CATALOG

Abbreviated entry: C(shift)A

Affected Basic abbreviations: None

Token: Hex $EE,$05 Decimal 238,5

Modes: Direct and program

Recommended mode: Direct

Purpose: To display the directory (CATALOG) of a disk in drive unit eight. This command will display the directory straight to the screen without having to load it in. Users of dual disk drives will be pleased to note that you can specify which drive to display by either a number one or zero after the command. If no number is specified,. the routine will default to drive zero.

Syntax: CATALOG [0 or 1]

Errors: Syntax error - if the command CATALOG is followed by anything but 0,1:, or nothing
Disk error message - after the CATALOG has been displayed, the disk error channel is read and displayed

Use: The command is used to display the directory of a disk. This can be useful if you have a program that you wish to save but need to check if there is room on the disk or find a filename to use. The directory can be paused when displaying by use of the spacebar, and restarted with any key. Display can be stopped completely with the STOP key.

Routine entry point: $85B6

Routine operation: On entry, the routine checks to see if a drive number is specified. If no number is specified or '0', the character '0' is inserted into the filename after the '$'. If it is a '1', the character '1' is inserted. Anything else will cause a Syntax error. The file is then opened and each line is read in and displayed ignoring line links. When the directory is finished, the file is closed and the disk error channel is read.

```
LOC    CODE          LINE


85B6                        .LIB CATALOG
85B6   F0 0B         CATLOG BEQ CATL01        ;DRIVE 0
85B8   C9 30                CMP #$30         ;IS IT 0?
85BA   F0 07                BEQ CATL01       ;YES
85BC   C9 31                CMP #$31         ;IS IT 1?
85BE   F0 06                BEQ CATL02       ;YES
85C0   4C 08 AF             JMP $AF08        ;SYNTAX ERROR
85C3   A9 30         CATL01 LDA #$30          ;CHAR '0'
85C5   2C                   .BYT $2C
85C6   A9 31         CATL02 LDA #$31          ;CHAR '1'
85C8   8D 83 86             STA OPDIR+1      ;STORE IN STRING
85CB   A9 02                LDA #$02         ;LENGTH
85CD   A2 82                LDX #<OPDIR      ;ADDRESS LSB
85CF   A0 86                LDY #>OPDIR      ;MSB
85D1   20 BD FF             JSR $FFBD        ;SET FILENAME DETAILS
85D4   A9 0E                LDA #$0E
85D6   20 A3 8A             JSR GETN1        ;GET UNUSED FILE#
85D9   A2 08                LDX #$08         ;DEVICE 8
85DB   A0 00                LDY #$00         ;SA 0
85DD   20 BA FF             JSR $FFBA        ;SET FILE DETAILS
85E0   20 C0 FF             JSR $FFC0        ;OPEN FILE
85E3   90 0A                BCC CATL03       ;NO ERROR
85E5   48                   PHA              ;STORE ERROR
85E6   A5 B8                LDA $B8          ;GET FILE #
85E8   20 C3 FF             JSR $FFC3        ;CLOSE FILE
85EB   68                   PLA              ;GET ERROR
85EC   4C F9 E0             JMP $E0F9        ;SEND ERROR
85EF                 ;
85EF   A0 03         CATL03 LDY #$03
85F1   84 B7         CATL04 STY $B7
85F3   A6 B8                LDX $B8
85F5   20 C6 FF             JSR $FFC6        ;SET INPUT DEVICE
85F8   20 CF FF             JSR $FFCF        ;INPUT
85FB   85 57                STA $57          ;STORE VALUE
85FD   20 B7 FF             JSR $FFB7        ;GET STATUS
8600   D0 72                BNE CATL13       ;STATUS ERROR
8602   20 CF FF             JSR $FFCF        ;INPUT
8605   85 58                STA $57+1        ;STORE IT
8607   20 B7 FF             JSR $FFB7        ;GET STATUS
860A   D0 68                BNE CATL13       ;STATUS ERROR
860C   A4 B7                LDY $B7          ;GET COUNTER
860E   88                   DEY              ;DO NEXT
860F   D0 E0         CATL05 BNE CATL04
8611   84 B7                STY $B7          ;SET $B7 TO ZERO
8613   20 CF FF      CATL06 JSR $FFCF         ;INPUT
8616   48                   PHA              ;STORE IT
8617   20 B7 FF             JSR $FFB7        ;GET STATUS
861A   AA                   TAX              ;STORE TO X
861B   68                   PLA              ;GET INPUT CHAR
861C   E0 00                CPX #$00         ;WAS THERE AN ERROR?
861E   D0 54                BNE CATL13       ;YES
8620   A4 B7                LDY $B7          ;GET LENGTH
```

```
LOC   CODE       LINE


8622  C0 50             CPY #$50        ;TOO LONG?
8624  B0 4E             BCS CATL13      ;YES, ERROR
8626  99 00 02          STA $0200,Y     ;STORE CHARACTER
8629  AA                TAX
862A  F0 04             BEQ CATL07      ;END OF LINE
862C  E6 B7             INC $B7         ;DO NEXT CHAR
862E  D0 E3             BNE CATL06      ;ALWAYS
8630             ;
8630  20 CC FF   CATL07 JSR $FFCC        ;RESET DEFAULT IO
8633  A6 9F             LDX $9F
8635  E0 03             CPX #$03
8637  F0 05             BEQ CATL08
8639  A6 9E             LDX $9E
863B  20 C9 FF          JSR $FFC9       ;SET OUTPUT DEVICE
863E  A6 57      CATL08 LDX $57
8640  A5 58             LDA $57+1
8642  20 CD BD          JSR $BDCD       ;PRINT FILE LENGTH
8645  A9 20             LDA #$20        ;SPACE CHAR
8647  20 D2 FF          JSR $FFD2       ;PRINT IT
864A  A0 00             LDY #$00
864C  B9 00 02   CATL09 LDA $0200,Y      ;GET CHAR
864F  F0 06             BEQ CATL10      ;END OF LINE
8651  20 D2 FF          JSR $FFD2       ;PRINT CHAR
8654  C8                INY
8655  D0 F5             BNE CATL09      ;DO NEXT LINE
8657  A9 0D      CATL10 LDA #$0D         ;CARRIAGE RETURN
8659  20 D2 FF          JSR $FFD2       ;PRINT IT
865C  20 CC FF          JSR $FFCC       ;RESET DEFAULT IO
865F  20 E1 FF          JSR $FFE1       ;STOP KEY?
8662  F0 10             BEQ CATL13      ;YES
8664  20 E4 FF          JSR $FFE4       ;GET KEY
8667  C9 20             CMP #$20        ;SPACE?
8669  D0 05             BNE CATL12      ;NO
866B  20 E4 FF   CATL11 JSR $FFE4        ;GET KEY
866E  F0 FB             BEQ CATL11      ;NO KEY
8670  A0 02      CATL12 LDY #$02
8672  D0 9B             BNE CATL05      ;DO NEXT LINE
8674  20 CC FF   CATL13 JSR $FFCC        ;RESET DEFAULT IO
8677  A5 B8             LDA $B8         ;GET FILE NUMBER
8679  20 C3 FF          JSR $FFC3       ;CLOSE FILE
867C  20 55 8A          JSR DISK01
867F  4C 74 A4          JMP $A474       ;JUMP TO READY VIA ERROR
8682  24 30      OPDIR  .BYT '$0'        ;FILE OPEN NAME
8684                    .END
```

CHAIN

Abbreviated entry: CHA(shift)I

Affected Basic abbreviations: None

Token: Hex $EE,$07   Decimal 238,7

Modes: Direct and program

Recommended mode: Either

Purpose: To load and run a Basic program from tape or disk. After the program has been loaded, variable pointers are set to the end of the program.

Syntax: As in LOAD

Errors: As in LOAD

Use: CHAIN is used to load and run a Basic program. It will work from another program or in direct mode, having the same effect. If used from another program, it is more convenient than LOAD as LOAD does not set the variable pointers, and if the program you load is larger than the one in memory, when variables are used they will corrupt the end of the program.

Routine entry point: $8684

Routine operation: The CHAIN routine simulates the LOAD routine as far as the program has been loaded. From there variable pointers are set to the end of LOAD, the run mode flag is set, and then three operations cause the program to run:

```
JSR $A65E  ;perform CLR
JSR $A68E  ;set charget pointers to the start of program
JMP $A7AE  ;execute NEXT command
```

```
LOC   CODE        LINE


8684                     .LIB CHAIN
8684  20 D4 E1    CHAIN  JSR $E1D4       ;GET NAME
8687  A9 00              LDA #$00
8689  85 B9              STA $B9         ;SECONDARY ADDRESS=0
868B  A6 2B              LDX $2B
868D  A4 2C              LDY $2C         ;ADDRESS TO LOAD AT
868F  20 D5 FF           JSR $FFD5       ;LOAD IT
8692  B0 21              BCS CHAIN1      ;LOAD WAS NOT O.K.
8694  86 2D              STX $2D         ;SAVE END OF LOAD
8696  86 2F              STX $2F         ; ADDRESS IN VARIABLE
8698  86 31              STX $31         ; POINTERS
869A  84 2E              STY $2E
869C  84 30              STY $30
869E  84 32              STY $32
86A0  A9 0D              LDA #$0D        ;PRINT CR
86A2  20 D2 FF           JSR $FFD2
86A5  A9 00              LDA #$00        ;SET TO RUN
86A7  85 9D              STA $9D
86A9  8D 24 9D           STA REPESK      ;CLEAR REPEAT STACK
86AC  20 5E A6           JSR $A65E       ;CLR
86AF  20 8E A6           JSR $A68E       ;SET CHARGET POINTER
86B2  4C AE A7           JMP $A7AE       ;RUN
86B5  4C F9 E0    CHAIN1 JMP $E0F9       ;SEND ERROR MESSAGE
86B8                     .END
```

CHANGE

Abbreviated entry: C(shift)H

Affected Basic abbreviations: CHR$ – CH(shift)R

Token: Hex $EE,$06 Decimal 238,6

Modes: Direct and program

Recommended mode: Direct only

Purpose: To change all occurrences of a string or command to something else. Each line that is changed is listed if there is anything left to list.

Syntax: CHANGE dstr1ddstr2d - where d is a delimiter character that does not appear in either of the strings (str1 or str2).

Errors: Syntax error - if the format is not as above
String too long - if either str1 or str2 are longer than 40 characters

Use: CHANGE has a number of uses. An example would be:

CHANGE @PRINT@@PRINT#4,@

to change all occurrences of PRINT to PRINT#4, or:

CHANGE "PRINT""PRINT#4,"

which will change all occurrences of the text PRINT to the text PRINT#4.

Note: Not all delimeter characters will work in all cases. An example is:

CHANGE /REM///

As the character '/' has two values, the first is the token for divide and the second is just the ASCII slash character.

The same is true of DATA. Other characters that will have the same effect are: '+-*↑=<>'.

Routine entry point: $86BB

Routine operation: CHANGE uses most of the FIND routines to find str1 and list the line.

CHANGE reads in the delimiter byte and stores it away. The string to be changed is then read in until the second delimiter character is reached and then stored away. The next character is checked to see that it equals the delimiter character, and if so the string to change to is read in until the delimiter character is found again or the end of command. The rest of the routine is just a loop finding all occurrences, changing them and listing until the end of the program.

The actual routine that changes the string uses the Basic input buffer and the Basic routines to change a line. The routine copies the line up to str1 into the buffer, the change string (str2) is then copied to the buffer and the remainder of the line is copied over. The pointers are then set so that the next byte to check is the one following str2.

```
LOC    CODE        LINE

86B8                      .LIB CHANGE
86B3   20 F3 84    CHANGE JSR RESVAR       ;RESET LINE LINKS
86BB   20 91 8E           JSR FIND14      ;GET CURRENT CHAR
```

```
LOC     CODE            LINE

86BE    85 59                   STA $59         ;STORE IN FLAG
86C0    A2 00                   LDX #$00
86C2    20 C7 8D                JSR FIND03      ;GET SEARCH STRING
86C5    A2 00                   LDX #$00
86C7    20 22 87                JSR CHAN07      ;GET STRING TO CHANGE
86CA    86 FC                   STX $FC         ;STORE LENGTH OF CHANGE STRING
86CC    20 E5 8D                JSR FIND05      ;SETUP POINTERS
86CF    78                      SEI
86D0    AD 00 03                LDA $0300
86D3    8D CF 8E                STA FINDER
86D6    AD 01 03                LDA $0301
86D9    8D D0 8E                STA FINDER+1
86DC    A9 67                   LDA #<FIND11    ;ERROR LINK TO RTS
86DE    8D 00 03                STA $0300
86E1    A9 8E                   LDA #>FIND11
86E3    8D 01 03                STA $0301
86E6    58                      CLI
86E7    20 F3 8D                JSR FIND06      ;FIND STRING
86EA    4C F6 86        CHAN01  JMP CHAN03       ;CHANGE
86ED    20 68 8E        CHAN02  JSR FIND12       ;LIST LINE
86F0    20 F9 8D                JSR FIND07      ;FIND STRING
86F3    4C EA 86                JMP CHAN01      ;AND REPEAT
86F6                    ;
86F6    A5 FC           CHAN03  LDA $FC          ;LENGTH OF CHANGE STRING
86F8    38                      SEC
86F9    E5 22                   SBC $22         ;- LENGTH OF FIND
86FB    F0 03                   BEQ CHAN04      ;THEY ARE EQUAL
86FD    4C 48 87                JMP CHAN10      ;ELSE CHANGE SIZE
8700    A4 23           CHAN04  LDY $23          ;INDEX TO LINE
8702    A2 40                   LDX #$40        ;INDEX TO CHANGE STRING
8704    A5 01                   LDA $01
8706    29 FE                   AND #$FE        ;OUT BASIC ROM
8708    85 01                   STA $01
870A    BD 40 BF        CHAN05  LDA $BF40,X      ;GET CHANGE CHAR
870D    F0 07                   BEQ CHAN06      ;END OF STRING
870F    91 57                   STA ($57),Y     ;REPLACE CHAR
8711    E8                      INX             ;NEXT CHAR
8712    C8                      INY             ;NEXT BYTE
8713    4C 0A 87                JMP CHAN05      ;AND AGAIN
8716    A5 01           CHAN06  LDA $01
8718    09 01                   ORA #$01        ;IN BASIC ROM
871A    85 01                   STA $01
871C    88                      DEY
871D    84 23                   STY $23         ;STORE LINE INDEX
871F    4C ED 86                JMP CHAN02      ;DO NEXT FIND
8722                    ;
8722    20 8B 8E        CHAN07  JSR FIND13       ;GET NEXT CHAR
8725    C5 59                   CMP $59         ;IS IT THE FLAG?
8727    F0 03                   BEQ CHAN08      ;YES, GET STRING
8729    4C 08 AF                JMP $AF08
872C    20 8B 8E        CHAN08  JSR FIND13       ;GET NEXT CHAR
872F    F0 11                   BEQ CHAN09      ;END OF LINE
8731    C5 59                   CMP $59         ;END OF STRING?
8733    F0 0D                   BEQ CHAN09      ;YES
8735    9D 80 BF                STA $BF80,X     ;STORE CHAR
8738    E8                      INX
8739    E0 40                   CPX #$40        ;STRING TOO LONG?
873B    D0 EF                   BNE CHAN08      ;NO
873D    A2 17                   LDX #$17        ;STRING TOO LONG
873F    4C 37 A4                JMP $A437       ;OUTPUT ERROR
8742    A9 00           CHAN09  LDA #$00         ;STRING TERMINATOR
8744    9D 80 BF                STA $BF80,X     ;STORE IT
8747    60                      RTS
8748                    ;
8748    A0 00           CHAN10  LDY #$00
```

```
LOC    CODE          LINE


874A   B1 57                LDA ($57),Y     ;GET LINE# LO
874C   85 14                STA $14         ;STORE IT
874E   C8                   INY
874F   B1 57                LDA ($57),Y     ;GET LINE# HI
8751   85 15                STA $15         ;STORE IT
8753   A2 00                LDX #$00
8755   C8            CHAN11 INY
8756   C4 23                CPY $23         ;REACHED STRING?
8758   F0 0A                BEQ CHAN12      ;YES, INSERT IT
875A   B1 57                LDA ($57),Y     ;GET PROGRAM BYTE
875C   9D 00 02             STA $0200,X     ;STORE IN BUFFER
875F   E8                   INX
8760   E0 56                CPX #$56        ;BUFFER TOO LARGE?
8762   D0 F1                BNE CHAN11      ;NOT YET
8764   A5 01         CHAN12 LDA $01
8766   29 FE                AND #$FE        ;OUT BASIC ROM
8768   85 01                STA $01
876A   A0 00                LDY #$00
876C   B9 80 BF      CHAN13 LDA $BF80,Y      ;GET CHANGE STRING BYTE
876F   F0 09                BEQ CHAN14      ;END OF STRING
8771   9D 00 02             STA $0200,X     ;STORE IN BUFFER
8774   E8                   INX             ;NEXT CHAR
8775   C8                   INY             ;AND PROGRAM BYTE
8776   E0 57                CPX #$57        ;END OF BUFFER?
8778   D0 F2                BNE CHAN13      ;NO
877A   A5 01         CHAN14 LDA $01
877C   09 01                ORA #$01        ;IN BASIC ROM
877E   85 01                STA $01
8780   A5 23                LDA $23         ;CALCULATE START
8782   18                   CLC             ;OF REST OF PROGRAM LINE
8783   65 22                ADC $22         ;AFTER INSERTING THE
8785   A8                   TAY             ;CHANGE STRING
8786   A5 23                LDA $23
8788   18                   CLC
8789   65 FC                ADC $FC
878B   85 23                STA $23
878D   C6 23                DEC $23
878F   B1 57         CHAN15 LDA ($57),Y      ;GET PROGRAM BYTE
8791   9D 00 02             STA $0200,X     ;STORE IN BUFFER
8794   C8                   INY             ;NEXT BYTE
8795   E8                   INX             ;NEXT CHAR
8796   C9 00                CMP #$00        ;END OF LINE?
8798   F0 0A                BEQ CHAN16      ;YES
879A   E0 58                CPX #$58        ;END OF BUFFER?
879C   D0 F1                BNE CHAN15      ;NOT YET
879E   A9 00                LDA #$00        ;ZERO IF END OF BUFFER
87A0   9D 00 02             STA $0200,X     ;STORE IT
87A3   E8                   INX
87A4   8E FC 87      CHAN16 STX CHANLN       ;STORE LENGTH OF
87A7   8A                   TXA             ;LINE
87A8   18                   CLC
87A9   69 04                ADC #$04
87AB   85 0B                STA $0B
87AD   AD 02 03             LDA $0302
87B0   8D FD 87             STA CHANST
87B3   AD 03 03             LDA $0303
87B6   8D FE 87             STA CHANST+1
87B9   A9 CB                LDA #<CHAN17    ;BASIC WARM START
87BB   8D 02 03             STA $0302       ;RE-ENTRY POINT
87BE   A9 87                LDA #>CHAN17
87C0   8D 03 03             STA $0303
87C3   20 96 8E             JSR FIND15      ;SAVE POINTERS ETC
87C6   A4 0B                LDY $0B         ;GET POINTER
87C8   4C A4 A4             JMP $A4A4       ;INSERT PROGRAM LINE
87CB   AD FD 87      CHAN17 LDA CHANST       ;RESTORE WARM START VECTOR
```

```
LOC   CODE        LINE


87CE  8D 02 03             STA $0302
87D1  AD FE 87             LDA CHANST+1
87D4  8D 03 03             STA $0303
87D7  20 B0 8E             JSR FIND16       ;RESTORE POINTERS ETC
87DA  A5 57                LDA $57          ;LAST LINE?
87DC  C5 2D                CMP $2D
87DE  D0 06                BNE CHAN18       ;NOT YET
87E0  A5 58                LDA $58
87E2  C5 2E                CMP $2E
87E4  F0 13                BEQ CHAN20       ;YES
87E6  AD FC 87    CHAN18   LDA CHANLN        ;DID WE DELETE
87E9  C9 01                CMP #$01         ;WHOLE LINE?
87EB  F0 03                BEQ CHAN19       ;YES
87ED  4C ED 86             JMP CHAN02       ;NO, LIST AND DO NEXT
87F0  A0 02       CHAN19   LDY #$02          ;INDEX TO NEXT LINE
87F2  84 23                STY $23
87F4  A2 00                LDX #$00
87F6  4C F0 86             JMP CHAN02+3     ;DO NEXT WITHOUT LIST
87F9  4C 56 8E    CHAN20   JMP FIND10        ;EXIT CHANGE
87FC  00          CHANLN   .BYT 0
87FD  00 00       CHANST   .WOR 0
87FF                       .END
```

CRUNCH

Abbreviated entry: C(shift)R

Affected Basic abbreviations: None

Token: Hex $EE,$08   Decimal 238,8

Modes: Direct and program

Recommended mode: Direct

Purpose: To remove all occurrences of REM in a program and so reduce the size of the program.

Syntax: CRUNCH

Errors: None

Use: CRUNCH is used to remove REM statements and anything following them on the same line. If the REM is in the first or second position of the line, a colon is left on the line in case there is a GOTO or GOSUB to that line.

Routine entry point: $87FF

Routine operation: The Basic program is scanned line by line, byte by byte, until the REM token is found. If REM is found in either the first or second byte of the program then a colon is put into the input buffer. Otherwise the whole line up to the REM is copied into the buffer and the Basic routine is used to alter the line. A '.' character is printed to tell you that it has found a REM token.

```
LOC   CODE          LINE


87FF                         .LIB CRUNCH
87FF  20 33 A5     CRUNCH JSR $A533       ;RE-CHAIN LINES
8802  A5 2B               LDA $2B        ;GET START OF BASIC
8804  85 FB               STA $FB        ; AND STORE IN TEMP
8806  A5 2C               LDA $2C        ; LOCATIONS FOR THE
8808  85 FC               STA $FC        ; USE OF THIS ROUTINE
880A                ;
880A  20 E1 FF     CRUN01 JSR $FFE1       ;STOP KEY?
880D  F0 09               BEQ CRUN02     ;YES
880F  A0 00               LDY #$00       ;MAIN CRUNCH LOOP
8811  B1 FB               LDA ($FB),Y    ;NEXT LINE LO
8813  C8                  INY
8814  11 FB               ORA ($FB),Y    ;NEXT LINE HI
8816  D0 03               BNE CRUN03     ;NOT END OF PROG
8818  4C 74 A4     CRUN02 JMP $A474       ;'READY.'
881B                ;
881B  C8           CRUN03 INY             ;GET LINE NUMBER
881C  B1 FB               LDA ($FB),Y    ;LO
881E  85 14               STA $14        ;LINE# LO
8820  C8                  INY
8821  B1 FB               LDA ($FB),Y    ;HI
8823  85 15               STA $15        ;LINE# HI
8825                                     ;
8825  C8           CRUN04 INY
8826  B1 FB               LDA ($FB),Y    ;NEXT BYTE OF LINE
8828  C9 8F               CMP #$8F       ;REM TOKEN?
882A  F0 28               BEQ CRUN09     ;YES, REMOVE REM
882C  C9 00               CMP #$00       ;END OF LINE?
882E  D0 0E               BNE CRUN06     ;NOT YET
8830  A0 00        CRUN05 LDY #$00        ;GET POINTERS TO
8832  B1 FB               LDA ($FB),Y    ; NEXT LINE AND
8834  AA                  TAX            ; STORE
8835  C8                  INY            ; IN POINTER
8836  B1 FB               LDA ($FB),Y
8838  85 FC               STA $FC
883A  86 FB               STX $FB
883C  D0 CC               BNE CRUN01     ;ALWAYS
883E                ;
883E  C9 EE        CRUN06 CMP #$EE        ;TOKEN?
8840  D0 03               BNE CRUN07     ;NO
8842  C8                  INY
8843  D0 E0               BNE CRUN04     ;SCAN OTHER HALF
8845  C9 22        CRUN07 CMP #$22        ;QUOTES?
8847  D0 DC               BNE CRUN04     ;NO
8849  C8           CRUN08 INY             ;YES, SCAN TO
884A  B1 FB               LDA ($FB),Y
884C  F0 E2               BEQ CRUN05     ; END OF LINE
884E  C9 22               CMP #$22       ; OR ANOTHER QUOTE?
8850  F0 D3               BEQ CRUN04     ;YES
8852  D0 F5               BNE CRUN08     ;NO, ALWAYS
8854                ;
8854  84 02        CRUN09 STY $02         ;STORE OFF INDEX TO LINE
8856  C0 06               CPY #$06       ;ON 1ST OR 2ND POS?
8858  90 10               BCC CRUN11     ;YES
885A  C6 02               DEC $02        ;COPY LINE TO REM INTO
885C  A0 04               LDY #$04       ; INPUT BUFFER
885E  B1 FB        CRUN10 LDA ($FB),Y
8860  99 FC 01            STA $01FC,Y
8863  C8                  INY
8864  C4 02               CPY $02        ;REACHED REM?
8866  D0 F6               BNE CRUN10     ;NO
8868  F0 07               BEQ CRUN12     ;ALWAYS, CHANGE
886A                ;
886A  A9 3A        CRUN11 LDA #$3A        ;PUT ':' AT START OF LINE
886C  8D 00 02            STA $0200      ; INTO INPUT BUFFER
```

```
LOC   CODE       LINE


886F  A0 05                  LDY #$05      ; AND INSERT IT
8871  A9 00      CRUN12 LDA #$00      ;SET ZERO TERMINATOR
8873  99 FC 01               STA $01FC,Y
8876  C8                     INY
8877  84 0B                  STY $0B
8879  A9 95                  LDA #<CRUN13  ;RETURN FROM CHANGE
887B  8D 02 03               STA $0302
887E  A9 88                  LDA #>CRUN13
8880  8D 03 03               STA $0303
8883  A5 FB                  LDA $FB       ;STORE LINE POINTER
8885  8D 34 03               STA $0334
8888  A5 FC                  LDA $FC
888A  8D 35 03               STA $0335
888D  A9 2E                  LDA #$2E      ;TELL USER WE ARE
888F  20 D2 FF               JSR $FFD2     ;DOING SOMETHING
8892  4C A4 A4               JMP $A4A4     ;CHANGE
8895             ;
8895  A9 83      CRUN13 LDA #$83      ;TO HERE FROM CHANGE
8897  8D 02 03               STA $0302     ;RESET WARM START
889A  A9 A4                  LDA #$A4      ; POINTER
889C  8D 03 03               STA $0303
889F  AD 34 03               LDA $0334     ;RESTORE LINE POINTER
88A2  85 FB                  STA $FB
88A4  AD 35 03               LDA $0335
88A7  85 FC                  STA $FC
88A9  D0 85                  BNE CRUN05    ;ALWAYS, NEXT LINE
88AB                         .END
```

CTL

Abbreviated entry: C(shift)T

Affected Basic abbreviations: None

Token: Hex $EE,$02 Decimal 238,2

Modes: Direct, program, and in PRINT statements

Purpose: To replace cursor and colour characters, screen and border pokes, thus improving the ability to position the cursor anywhere on the screen. If the value is not specified, the current value is used.

Syntax: CTL ([x][,[y][,[cc][,[sc][,[bc][,[cls]]]]]]) – where x is the column position of the cursor (0–39), y is the row position of the cursor (0–24), cc is the cursor colour, sc the screen colour, bc is the border colour (0–15), and cls is a flag for clearing the screen (0 = no, 1 = yes).

Errors: Syntax error – if the syntax is not as above
Illegal quantity – if the values are out of range

Use: CTL is a powerful screen handling routine. Cursor, screen, and border colours can be set with a number (0–15), and the position of the cursor on the screen can be anywhere you like by entering the x position (0–39) and the y position (0–24). There is also a screen clear flag which, if set to 1, will clear the screen before positioning the cursor. To make it easier to describe, here are a few examples with details of what they do.

CTL (20) – positions the cursor at the middle of the current line
CTL (0,0) – moves the cursor to 0,0 (home position)
CTL (,,1) – sets the cursor color to white
CTL (,,,0) – sets the screen colour to black
CTL (,,,,0) – sets the border colour to black
CTL (,,,,,1) – clears the screen leaving the cursor at the current position
CTL (20,12,5,0,11,1) – clears the screen (1), sets the screen to black (0), the border to medium grey (11), the cursor colour to green (5), and the cursor position to column 20, row 12.

To print something at a specified location on the screen:

PRINT CTL(x,y)"text"CTL(x1,y1)"more text".....

Routine entry point: $88AB

Routine operation: The current settings of the five parameters are read and the screen clear flag is set to 0. The open brackets character is scanned past and each of the six values is read if present, checking to see if there is a closing bracket. When the closing bracket is found the screen is cleared if the flag is set to 1, and the other values are stored in their own locations.

```
LOC   CODE       LINE


88AB                    .LIB CTL
88AB  20 2C 89   CTL    JSR CTLDEF      ;SET DEFAULT
88AE  20 FA AE          JSR $AEFA      ;SCAN '('
88B1  20 79 00          JSR $0079      ;GET CURRENT CHAR
88B4  20 51 89          JSR CHECKN+3   ;NEXT PAR?
89B7  B0 08             BCS CTL01      ;NO
88B9  20 65 89          JSR GV1        ;GET VALUE
88BC  8E 8F 89          STX CTXPOS     ;STORE IT
88BF  B0 42             BCS CTLEN1     ;FOLLOWED BY ')'
88C1  20 4E 89   CTL01  JSR CHECKN      ;NEXT PAR?
88C4  B0 08             BCS CTL02      ;NO
88C6  20 6E 89          JSR GV4        ;GET VALUE
88C9  8E 90 89          STX CTYPOS     ;STORE IT
88CC  B0 35             BCS CTLEN1     ;FOLLOWED BY ')'
88CE  20 4E 89   CTL02  JSR CHECKN      ;NEXT PAR?
88D1  B0 08             BCS CTL03      ;NO
88D3  20 68 89          JSR GV2        ;GET VALUE
88D6  8E 91 89          STX CTCUR      ;STORE IT
88D9  B0 28             BCS CTLEN1     ;FOLLOWED BY ')'
88DB  20 4E 89   CTL03  JSR CHECKN      ;NEXT PAR?
88DE  B0 08             BCS CTL04      ;NO
88E0  20 68 89          JSR GV2        ;GET VALUE
88E3  8E 92 89          STX CTSC       ;STORE IT
88E6  B0 1B             BCS CTLEN1     ;FOLLOWED BY ')'
88E8  20 4E 89   CTL04  JSR CHECKN      ;NEXT PAR?
88EB  B0 08             BCS CTL05      ;NO
88ED  20 68 89          JSR GV2        ;GET VALUE
88F0  8E 93 89          STX CTBD       ;STORE IT
88F3  B0 0E             BCS CTLEN1     ;FOLLOWED BY ')'
```

```
LOC   CODE        LINE


88F5  20 4E 89    CTL05  JSR CHECKN      ;NEXT PAR?
88F8  90 03              BCC CTL06      ;YES
88FA  4C 08 AF           JMP $AF08      ;COMMA, SYNTAX ERROR
88FD  20 6B 89    CTL06  JSR GV3         ;GET VALUE
8900  8E 94 89           STX CTCFLG     ;STORE IT
8903  20 F7 AE    CTLEN1 JSR $AEF7       ;SCAN ')'
8906              ;
8906  AD 94 89    CTLEND LDA CTCFLG      ;CLEAR SCREEN?
8909  F0 05              BEQ CTEND1     ;NO
890B  A9 93              LDA #147       ;CHAR FOR CLS
890D  20 16 E7           JSR $E716      ;OUTPUT TO SCREEN
8910  AD 91 89    CTEND1 LDA CTCUR       ;GET CURSOR COLOUR
8913  8D 86 02           STA $0286      ;SET IT
8916  AD 92 89           LDA CTSC       ;GET SCREEN COLOUR
8919  8D 21 D0           STA $D021      ;SET IT
891C  AD 93 89           LDA CTBD       ;GET BORDER COLOUR
891F  8D 20 D0           STA $D020      ;SET IT
8922  AC 8F 89           LDY CTXPOS     ;GET X POSITION
8925  AE 90 89           LDX CTYPOS     ;GET Y POSITION
8928  18                 CLC            ;FLAG WRITE
8929  4C F0 FF           JMP $FFF0      ;SET CURSOR POS AND EXIT
892C              ;
892C  38          CTLDEF SEC             ;FLAG READ
892D  20 F0 FF           JSR $FFF0      ;GET CURSOR POS
8930  8C 8F 89           STY CTXPOS     ;STORE X
8933  8E 90 89           STX CTYPOS     ;STORE Y
8936  AD 21 D0           LDA $D021      ;GET SCREEN COLOUR
8939  8D 92 89           STA CTSC       ;STORE IT
893C  AD 20 D0           LDA $D020      ;GET BORDER COLOUR
893F  8D 93 89           STA CTBD       ;STORE IT
8942  AD 86 02           LDA $0286      ;GET CURSOR COLOUR
8945  8D 91 89           STA CTCUR      ;STORE IT
8948  A9 00              LDA #$00       ;ZERO SCREEN CLEAR
894A  8D 94 89           STA CTCFLG     ;FLAG
894D  60                 RTS
894E              ;
894E  20 73 00    CHECKN JSR $0073       ;GET NEXT CHAR
8951  C9 2C              CMP #$2C       ;IS IT A COMMA?
8953  D0 02              BNE CHECKB     ;NO
8955  38          CHECKS SEC             ;FLAG FOR COMMA
8956  60                 RTS
8957  C9 29       CHECKB CMP #$29        ;IS IT ')'?
8959  F0 02              BEQ CHECKA     ;YES, DONE
895B  18          CHECKC CLC             ;SET NO COMMA
895C  60                 RTS
895D  68          CHECKA PLA             ;REMOVE RTS
895E  68                 PLA            ;ADDRESS
895F  20 73 00           JSR $0073      ;GET NEXT CHAR
8962  4C 06 89           JMP CTLEND     ;SET VALUES
8965              ;
8965  A9 28       GV1    LDA #40         ;COMPARE X POS
8967  2C                 .BYT $2C       ;SKIP
8968  A9 10       GV2    LDA #16         ;COMPARE COLOUR
896A  2C                 .BYT $2C       ;SKIP
896B  A9 02       GV3    LDA #2          ;COMPARE CLEAR FLAG
896D  2C                 .BYT $2C       ;SKIP
896E  A9 19       GV4    LDA #25         ;COMPARE Y POS
8970  8D 8E 89           STA VCOMP      ;STORE COMPARE VALUE
8973  20 9E B7           JSR $B79E      ;GET 1 BYTE#
8976  EC 8E 89           CPX VCOMP      ;IN RANGE 0-(VCOMP-1)
8979  B0 0E              BCS GERR       ;NO
897B  20 79 00           JSR $0079      ;GET CURRENT CHAR
897E  C9 29              CMP #$29       ;IS IT ')'
8980  F0 D3              BEQ CHECKS     ;YES, FLAG END
8982  C9 2C              CMP #$2C       ;IS IT ','
```

```
LOC   CODE        LINE

8984  F0 D5              BEQ CHECKC      ;YES FLAG ANOTHER
8986  4C 08 AF           JMP $AF08       ;SYNTAX ERROR
8989  A2 0E       GERR   LDX #$0E         ;ILLEGAL QUANTITY
898B  4C 37 A4           JMP $A437       ;SEND ERROR
898E              ;
898E  00          VCOMP  .BYT 0           ;VALUE COMPARE
898F  00          CTXPOS .BYT 0           ;X POSITION
8990  00          CTYPOS .BYT 0           ;Y POSITION
8991  00          CTCUR  .BYT 0           ;CURSOR COLOUR
8992  00          CTSC   .BYT 0           ;SCREEN COLOUR
8993  00          CTBD   .BYT 0           ;BORDER COLOUR
8994  00          CTCFLG .BYT 0           ;CLEAR SCREEN FLAG
8995                     .END
```

DEEK

Abbreviated entry: D(shift)E

Affected Basic abbreviations: DEF - DEF

Token: Hex $EE,$1D Decimal 238,29

Modes: Direct and program

Recommended mode: Either

Purpose: To return the value of a two byte pointer that is stored lo,hi order.

Syntax: DEEK (expression) - where expression is the address of the low byte of the number.

Errors: Syntax error
Illegal quantity - if the expression is less than 0 or greater than 65535

Use: DEEK stands for Double byte pEEK and is used to get a two byte value stored in the 6510 microprocessor's internal two byte format, e.g.

DEEK(43) - returns the beginning of Basic
PEEK(43)+PEEK(44)*256 - is the normal way of getting the value

Note: DEEK must be on the right-hand side of an expression e.g. B=DEEK(43) and not DEEK(43)=B.

Routine entry point: $8995

Routine operation: The two byte address inside the brackets is read in and stored in $14,$15. Using this value the bytes are read and converted to floating point form.

```
LOC    CODE         LINE


8995                        .LIB DEEK
8995   20 8A AD     DEEK    JSR $AD8A        ;GET NUMBER
8998   20 F7 B7             JSR $B7F7       ;MAKE INTEGER
899B   A0 00                LDY #$00
899D   B1 14                LDA ($14),Y     ;GET LO BYTE
899F   AA                   TAX             ;INTO .X
89A0   C8                   INY
89A1   B1 14                LDA ($14),Y     ;GET HI BYTE
89A3   86 63        ASSIGN  STX $63          ;STORE LO BYTE
89A5   85 62                STA $62         ;STORE HI BYTE
89A7   A2 90                LDX #$90        ;EXPONENT =$90
89A9   38                   SEC
89AA   4C 49 BC             JMP $BC49       ;FLOAT AND SEND
89AD                        .END
```

DELETE

Abbreviated entry: DE(shift)L

Affected Basic abbreviations: None

Token: Hex $EE,$09 Decimal 238,9

Modes: Direct and program

Recommended mode: Direct only

Purpose: To delete a range of unwanted lines from a Basic program.

Syntax: DELETE [start line][−[end line]]. Although all parameters are denoted as optional, at least one of the parameters must be given.

Errors: Syntax error – if DELETE is used without parameters
Syntax error – if either of the line numbers is less than 0 or greater than 63999

Use: DELETE is used to delete a range of lines in a Basic program. These can be lines of, say, a data generating program after the DATA has been created. For example:

DELETE 100–150 – deletes lines 100 to 150 inclusive
DELETE −1000 – deletes all lines up to line number 1000
DELETE 2000– – deletes all lines from 2000 to the end of the program
DELETE 0 – deletes the whole program

Program lines that have been DELETEd can not be recovered as they have been wiped from memory.

Routine entry point: $89AD

Routine operation: DELETE first gets the range of the delete and then loops, moving the memory above the range over the top of the deleted area.

```
LOC   CODE        LINE


89AD                      .LIB DELETE
89AD  20 11 8A    DELETE  JSR DELE05       ;GET DELETE RANGE
89B0  A5 5F               LDA $5F         ;GET START OF DELETE
89B2  A6 60               LDX $5F+1       ;MEMORY POINTER
89B4  85 FB               STA $FB         ;STORE IT
89B6  86 FC               STX $FB+1
89B8  20 13 A6            JSR $A613       ;FIND ADDRESS OF
89BB  A5 5F               LDA $5F         ;END OF DELETE
89BD  A6 60               LDX $5F+1
89BF  90 0A               BCC DELE01
89C1  A0 01               LDY #$01
89C3  B1 5F               LDA ($5F),Y
89C5  F0 04               BEQ DELE01
89C7  AA                  TAX
89C8  88                  DEY
89C9  B1 5F               LDA ($5F),Y
89CB                      ;
89CB                      ;.A ,.X HOLD THE POINTER TO THE END
89CB                      ; OF DELETE RANGE.
89CB                      ;$FC,$FB HOLD THE POINTER TO THE
89CB                      ; START OF DELETE RANGE.
89CB                      ;
89CB  85 7A       DELE01  STA $7A          ;STORE AWAY END
89CD  86 7B               STX $7A+1       ;OF DELETE POINTER
89CF  A5 FB               LDA $FB
89D1  38                  SEC
89D2  E5 7A               SBC $7A         ;SET VARIABLE POINTER
89D4  AA                  TAX             ;TO END OF PROGRAM AFTER
89D5  A5 FC               LDA $FB+1       ;DELETE
89D7  E5 7B               SBC $7A+1
89D9  A8                  TAY
89DA  B0 1E               BCS DELE03
89DC  8A                  TXA
89DD  18                  CLC
89DE  65 2D               ADC $2D
89E0  85 2D               STA $2D
89E2  98                  TYA
89E3  65 2E               ADC $2D+1
89E5  85 2E               STA $2D+1
89E7  A0 00               LDY #$00
89E9  B1 7A       DELE02  LDA ($7A),Y      ;GET BYTE
89EB  91 FB               STA ($FB),Y     ;MOVE IT DOWN
89ED  C8                  INY
89EE  D0 F9               BNE DELE02      ;DO FULL PAGE
89F0  E6 7B               INC $7A+1       ;INCREMENT HI BYTE
89F2  E6 FC               INC $FB+1       ;POINTERS
89F4  A5 2E               LDA $2D+1       ;DONE LENGTH?
89F6  C5 FC               CMP $FB+1
89F8  B0 EF               BCS DELE02      ;NOT YET
89FA  20 33 A5    DELE03  JSR $A533        ;RE-CHAIN PROG
89FD  A5 2D               LDA $2D
89FF  A6 2E               LDX $2E
8A01  18                  CLC
8A02  69 02               ADC #$02
8A04  85 2D               STA $2D         ;SET VAR POINTERS
8A06  90 01               BCC DELE04
8A08  E8                  INX
8A09  86 2E       DELE04  STX $2D+1
8A0B  20 59 A6            JSR $A659       ;PERFORM 'CLR'
8A0E  4C 74 A4            JMP $A474       ;'READY.'
8A11                      ;
```

```
LOC    CODE          LINE


8A11                 ;GET RANGE FOR DELETE
8A11                 ;
8A11   20 79 00      DELE05 JSR $0079         ;GET CURRENT CHAR
8A14   F0 10                BEQ DELE06       ;NO RANGE, ERROR
8A16   90 11                BCC DELE07       ;IS A NUMBER
8A18   C9 AB                CMP #$AB         ;IS IT '-'?
8A1A   D0 0A                BNE DELE06       ;NO, ERROR
8A1C   A5 2B                LDA $2B          ;SET START ADDRESS OF
8A1E   85 5F                STA $5F          ;DELETE TO START
8A20   A5 2C                LDA $2C          ;OF PROGRAM
8A22   85 60                STA $5F+1
8A24   D0 12                BNE DELE08       ;ALWAYS
8A26   4C 08 AF      DELE06 JMP $AF08         ;OUTPUT SYNTAX ERROR
8A29   20 6B A9      DELE07 JSR $A96B         ;GET NUMBER
8A2C   20 13 A6             JSR $A613        ;FIND ADDRESS OF LINE
8A2F   20 79 00             JSR $0079        ;SECOND VALUE?
8A32   F0 0C                BEQ DELE09       ;NO
8A34   C9 AB                CMP #$AB         ;IS IT '-'?
8A36   D0 EE                BNE DELE06       ;NO, ERROR
8A38   20 73 00      DELE08 JSR $0073         ;GET NEXT CHAR
8A3B   20 6B A9             JSR $A96B        ;GET NUMBER
8A3E   D0 E6                BNE DELE06       ;NOT END OF INPUT
8A40   A5 14         DELE09 LDA $14           ;IS SECOND LINE ZERO?
8A42   05 15                ORA $14+1
8A44   D0 06                BNE DELE10       ;NO
8A46   A9 FF                LDA #$FF         ;SET TO MAX LINE#
8A48   85 14                STA $14
8A4A   85 15                STA $14+1
8A4C   60            DELE10 RTS               ;RANGE DONE
8A4D                        .END
```

DISK

Abbreviated entry: D(shift)I

Affected Basic abbreviations: DIM - DIM

Token: Hex $EE,$0A   Decimal 238,10

Modes: Direct and program

Recommended mode: Either

Purpose: To send a disk command to the disk unit eight.

Syntax: DISK [string expression] - where the string expression is:

“S0:TEST” - to scratch the file test
‘N0:DISK,00” - to reformat the entire disk

The other syntax is DISK which will display the disk error message to the screen giving a message like:

23,READ ERROR,18,01

where 23 is the error number, 18 is the track, 01 is the sector, and READ ERROR is the error description.

Errors: Syntax error - if the first character of the command is not a quote character
String too long - if the command is over 255 bytes long
Type mismatch - if the command is a number, not a string

Use: This command is useful in checking errors created from disk access by using just DISK which displays the message. A Basic equivalent would be:

OPEN 15,8,15
INPUT #15,E,EM$,T,S
PRINT E“,“EM$”,“T”,“S”,”
CLOSE 15

Also, for sending disk commands such as Scratch a file etc.:

DISK “10”

is equivalent to:

OPEN 15,8,15,“I0”

For disk commands refer to the disk user manual.

Routine entry point: $8A4D

Routine operation: The DISK routine checks to see if anything follows the command; if not the error channel is read and displayed. If there is text after the command (which must start with the quotes character) the text is read in and sent in the open command. Before either of these two operations is actioned, the current file is closed.

```
LOC   CODE         LINE


8A4D                       .LIB DISK
8A4D  20 79 00     DISK    JSR $0079         ; CHECK FOR BLANK
8A50  F0 03                BEQ DISK01       ; AFTER COMMAND.
8A52  4C B8 8A             JMP DISK04
8A55  A9 00        DISK01  LDA #$00          ; IF BLANK, READ
8A57  85 B7                STA $B7          ; ERROR MESSAGE
8A59  20 91 8A             JSR FOPEN        ; OPEN A FILE
8A5C  A9 0D                LDA #$0D         ; PRINT <RETURN>
8A5E  20 D2 FF             JSR $FFD2
8A61  A9 12                LDA #$12         ; PRINT <REVERSE ON>
8A63  20 D2 FF             JSR $FFD2
8A66  A6 B8                LDX $B8
8A68  20 C6 FF             JSR $FFC6        ; SET FILE TO INPUT
8A6B  20 CF FF     DISK02  JSR $FFCF         ; INPUT
8A6E  48                   PHA
8A6F  A5 90                LDA $90          ; CHECK STATUS
8A71  D0 07                BNE DISK03
8A73  68                   PLA
8A74  20 D2 FF             JSR $FFD2        ; PRINT CHARACTER
8A77  4C 6B 8A             JMP DISK02       ; AND NEXT
8A7A  68           DISK03  PLA
8A7B  A5 B8                LDA $B8
8A7D  85 49                STA $49
8A7F  20 CC E1             JSR $E1CC        ; CLOSE FILE
8A82  A9 92                LDA #$92
8A84  20 D2 FF             JSR $FFD2        ; PRINT <REVERSE OFF>
8A87  A9 0D                LDA #$0D
```

```
LOC    CODE          LINE

8A89   20 D2 FF             JSR $FFD2      ; PRINT <RETURN>
8A8C   A9 00                LDA #$00
8A8E   4C C6 FF             JMP $FFC6      ; INPUT TO KYBD
8A91                 ;
8A91   20 A1 8A      FOPEN  JSR GETFNO      ; FIND FREE FILE NO.
8A94   85 B8                STA $B8
8A96   A9 0F                LDA #$0F       ; SECONDARY ADDRESS
8A98   85 B9                STA $B9
8A9A   A9 08                LDA #$08       ; DEVICE NUMBER
8A9C   85 BA                STA $BA
8A9E   4C C1 E1             JMP $E1C1      ; OPEN
8AA1                 ;
8AA1   A9 0F         GETFNO LDA #$0F        ; CHECK TABLE OF
8AA3   A6 98         GETN1  LDX $98         ; FILE NUMBERS FOR
8AA5   E0 00                CPX #$00       ; A FREE ONE
8AA7   F0 0E                BEQ GETN4      ; HAS BEEN FOUND
8AA9   DD 58 02      GETN2  CMP $0258,X
8AAC   D0 06                BNE GETN3
8AAE   38                   SEC
8AAF   E9 01                SBC #$01
8AB1   4C A3 8A             JMP GETN1
8AB4   CA            GETN3  DEX             ; TRY NEXT NUMBER
8AB5   D0 F2                BNE GETN2
8AB7   60            GETN4  RTS
8AB8                 ;
8AB8   C9 22         DISK04 CMP #$22        ; CHECK  FOR COMMAND
8ABA   F0 03                BEQ DISK05     ; IN QUOTES
8ABC   4C 08 AF             JMP $AF08      ; SYNTAX ERROR
8ABF   A5 B8         DISK05 LDA $B8         ; CLOSE CURRENT
8AC1   85 49                STA $49        ; DISK FILE
8AC3   20 CC E1             JSR $E1CC
8AC6   20 9E AD             JSR $AD9E      ; GET TEXT IN QUOTES
8AC9   20 A3 B6             JSR $B6A3
8ACC   A6 22                LDX $22        ; STRING ADDRESS AT
8ACE   86 BB                STX $BB        ; ($22)
8AD0   A4 23                LDY $23
8AD2   84 BC                STY $BC
8AD4   85 B7         DISK07 STA $B7         ; SET LENGTH
8AD6   20 91 8A             JSR FOPEN      ; OPEN FILE
8AD9   A9 0D                LDA #$0D
8ADB   20 D2 FF             JSR $FFD2      ; PRINT <RETURN>
8ADE   60                   RTS            ; EXIT DISK
8ADF                        .END
```

DOKE

Abbreviated entry: D(shift)O

Affected Basic abbreviations: None

Token: Hex $EE,$0B   Decimal 238,11

Modes: Direct and program

Recommended mode: Either

Purpose: To store a value (0–65535) in the 6510 microprocessor's internal two byte format (the opposite of DEEK).

Syntax: DOKE address, value – where the address and value are between 0 and 65535.

Errors: Syntax error – if either of the values is out of the range 0-65535

Use: DOKE stores a two byte value into memory at the location pointed to by the address. It can be used for storing a frequency value to the SID chip:

DOKE 54272,10000

POKE 54272,INT(10000/256)
POKE 54273,10000-INT(10000/256)*256

Routine entry point: $8ADF

Routine operation: The two byte address is read in and stored to a safe location. The two byte value is then read in and the two bytes are stored in lo,hi order pointed to by the address.

```
LOC   CODE         LINE


8ADF                       .LIB DOKE
8ADF  20 8A AD     DOKE    JSR $AD8A       ;GET ADDRESS
8AE2  20 F7 B7             JSR $B7F7       ;CONVERT TO INT
8AE5  A5 14                LDA $14         ;GET LSB
8AE7  85 FB                STA $FB         ;SAVE IT
8AE9  A5 15                LDA $15         ;GET MSB
8AEB  85 FC                STA $FC         ;SAVE IT
8AED  20 FD AE             JSR $AEFD       ;SCAN PAST ','
8AF0  20 8A AD             JSR $AD8A       ;GET VALUE
8AF3  20 F7 B7             JSR $B7F7       ;CONVERT TO INT
8AF6  A0 00                LDY #$00        ;INDEX
8AF8  A5 14                LDA $14         ;GET LSB
8AFA  91 FB                STA ($FB),Y     ;STORE LSB
8AFC  C8                   INY             ;NEXT BYTE
8AFD  A5 15                LDA $15         ;GET MSB
8AFF  91 FB                STA ($FB),Y     ;STORE MSB
8B01  60                   RTS
8B02                       .END
```

DUMP

Abbreviated entry: D(shift)U

Affected Basic abbreviations: None

Token: Hex $EE,$0C   Decimal 238,12

Modes: Direct and program

Recommended mode: Direct

Purpose: To display the values of all simple variables, name functions, and display the dimensions of arrays.

Syntax: DUMP

Errors: None

Use: For de-bugging Basic programs, the DUMP command may be used after the program has run to get a list of all variables and their values. As an added bonus, not found in any other DUMP command for the Commodore 64, all array dimensions are also given. The DUMP command will also display function names.

Routine entry point: $8B02

Routine operation: The DUMP routine sets a pointer to the start of variables and checks for the end of variables. If it does not find any the variable name is read in and displayed, the variable type is determined, and the display is produced according to which type is required. When all simple variables have been processed, arrays are handled. The array names are read and displayed in the same way as the simple variables, and the number of dimensions read off. The pointer is then set to the end of the dimension entries and, reading backwards, the dimensions are read and displayed.

```
LOC   CODE        LINE


8B02                      .LIB DUMP
8B02  A5 2E       DUMP    LDA $2E         ;GET START OF VARIABLES
8B04  85 60               STA $60        ; AND STORE IN REQUIRED
8B06  A5 2D               LDA $2D        ; LOCATIONS
8B08  85 5F               STA $5F
8B0A              ;
8B0A  38          DUMP01  SEC             ;START OF MAIN LOOP
8B0B  E5 2F               SBC $2F        ;END OF VARIABLES?
8B0D  A5 60               LDA $60
8B0F  E5 30               SBC $30
8B11  90 03               BCC DUMP02     ;NO
8B13  4C ED 8B            JMP DUMP17     ;YES, DISPLAY ARRAY DIMS
8B16              ;
8B16  20 C5 8B    DUMP02  JSR DUMP12      ;GET VAR NAME
8B19  A5 25               LDA $25        ;REAL?
8B1B  F0 2B               BEQ DUMP03     ;YES
8B1D  C9 01               CMP #$01       ;FUNCTION?
8B1F  F0 47               BEQ DUMP26     ;YES
8B21  C9 02               CMP #$02       ;STRING?
8B23  F0 5C               BEQ DUMP04     ;YES
8B25  A9 25               LDA #$25       ;MUST BE INTEGER
8B27  20 D2 FF            JSR $FFD2      ;PRINT '%'
8B2A  20 E3 8B            JSR DUMP15     ;ANY EXTRA SPACES
8B2D  A9 3D               LDA #$3D
8B2F  20 D2 FF            JSR $FFD2      ;PRINT '='
8B32  A0 02               LDY #$02       ;SET POINTER TO VAL
8B34  B1 5F               LDA ($5F),Y    ;GET LO
8B36  48                  PHA
8B37  C8                  INY
8B38  B1 5F               LDA ($5F),Y    ;GET HI
8B3A  A8                  TAY
8B3B  68                  PLA
8B3C  20 91 B3            JSR $B391      ;FIX-FLOAT
8B3F  20 DD BD            JSR $BDDD      ;FLOAT-ASCII
8B42  20 1E AB            JSR $AB1E      ;PRINT IT
8B45  4C A9 8B            JMP DUMP07     ;DO NEXT VAR
8B48              ;
```

```
LOC     CODE            LINE


8B48                    ;REAL VARIABLE
8B48                    ;
8B48    A9 20           DUMP03 LDA #$20
8B4A    20 D2 FF               JSR $FFD2       ;PRINT SPACE
8B4D    20 E3 8B               JSR DUMP15      ;PAD NAME
8B50    A9 3D                  LDA #$3D
8B52    20 D2 FF               JSR $FFD2       ;PRINT '='
8B55    20 85 B1               JSR $B185       ;GET ADDRESS OF VAR
8B58    A5 47                  LDA $47         ; INTO A AND Y
8B5A    A4 48                  LDY $48
8B5C    20 A2 BB               JSR $BBA2       ;MEM-FAC#1
8B5F    20 DD BD               JSR $BDDD       ;FLOAT-ASCII
8B62    20 DA BD               JSR $BDDA       ;PRINT NUMBER
8B65    4C A9 8B               JMP DUMP07      ;DO NEXT VAR
8B68                    ;
8B68                    ;FUNCTION
8B68                    ;
8B68    20 E3 8B        DUMP26 JSR DUMP15       ;PAD NAME
8B6B    A9 75                  LDA #<FUNCTT    ;POINT TO
8B6D    A0 8B                  LDY #>FUNCTT    ;'FUNCTION'
8B6F    20 1E AB               JSR $AB1E       ;PRINT STRING
8B72    4C A9 8B               JMP DUMP07      ;DO NEXT VAR
8B75    20 3D           FUNCTT .BYT ' = FUNCTION',$00
8B80    00
8B81                    ;
8B81                    ;STRING VARIABLE
8B81                    ;
8B81    A2 03           DUMP04 LDX #$03         ;LOOP TO PRINT '$= "'
8B83    BD CA 8C        DUMP05 LDA DUMTBL,X
8B86    20 D2 FF               JSR $FFD2
8B89    E0 03                  CPX #$03
8B8B    D0 03                  BNE DUMP06
8B8D    20 E3 8B               JSR DUMP15      ;PAD FOR NAME
8B90    CA              DUMP06 DEX
8B91    10 F0                  BPL DUMP05      ;COMPLETE LOOP
8B93    A0 04                  LDY #$04        ;GET ADDRESS OF STRING
8B95    B1 5F                  LDA ($5F),Y
8B97    85 23                  STA $23
8B99    88                     DEY
8B9A    B1 5F                  LDA ($5F),Y
8B9C    85 22                  STA $22
8B9E    88                     DEY
8B9F    B1 5F                  LDA ($5F),Y     ;LENGTH
8BA1    20 24 AB               JSR $AB24       ;PRINT STRING FROM ($22)
8BA4    A9 22                  LDA #$22        ; AND LENGTH IN .A
8BA6    20 D2 FF               JSR $FFD2       ;PRINT '"'
8BA9                    ;
8BA9                    ;PRINT CARRIAGE RETURN AND DO NEXT
8BA9                    ;
8BA9    A9 0D           DUMP07 LDA #$0D
8BAB    20 D2 FF               JSR $FFD2       ;PRINT RETURN
8BAE    20 E1 FF        DUMP08 JSR $FFE1        ;STOP KEY?
8BB1    D0 01                  BNE DUMP10      ;NO
8BB3    60              DUMP09 RTS              ;EXIT TO 'READY'
8BB4    18              DUMP10 CLC              ;MOVE TO NEXT VAR
8BB5    A5 5F                  LDA $5F
8BB7    69 07                  ADC #$07
8BB9    85 5F                  STA $5F
8BBB    A6 60                  LDX $60
8BBD    90 01                  BCC DUMP11
8BBF    E8                     INX
8BC0    86 60           DUMP11 STX $60
8BC2    4C 0A 8B               JMP DUMP01      ;DO NEXT VAR
8BC5                    ;
8BC5                    ;GET AND PRINT VAR NAME
8BC5                    ;
```

```
LOC   CODE        LINE


8BC5  A0 00       DUMP12 LDY #$00        ;GET VARIABLE TYPE
8BC7  84 25              STY $25        ;AND NAME
8BC9  C8                 INY
8BCA  B1 5F       DUMP13 LDA ($5F),Y     ;GET BYTE
8BCC  0A                 ASL A          ;TYPE BIT INTO TEMP
8BCD  26 25              ROL $25
8BCF  4A                 LSR A          ;RESTORE NAME BYTE
8BD0  99 45 00           STA $0045,Y    ;STORE NAME BYTE
8BD3  88                 DEY
8BD4  10 F4              BPL DUMP13
8BD6  A5 45              LDA $45        ;PRINT NAME
8BD8  20 D2 FF           JSR $FFD2
8BDB  A5 46              LDA $46        ;2ND BYTE?
8BDD  F0 03              BEQ DUMP14     ;NO
8BDF  20 D2 FF           JSR $FFD2      ;YES, PRINT IT
8BE2  60          DUMP14 RTS             ;DONE
8BE3              ;
8BE3              ;PAD OUT NAME IF ONLY 1 BYTE LONG
8BE3              ;
8BE3  A5 46       DUMP15 LDA $46         ;2ND BYTE?
8BE5  D0 05              BNE DUMP16     ;YES, DON'T PAD
8BE7  A9 20              LDA #$20       ;ELSE PAD WITH SPACE
8BE9  20 D2 FF           JSR $FFD2      ;PRINT
8BEC  60          DUMP16 RTS             ;DONE
8BED              ;
8BED              ;DISPLAY ARRAY NAMES AND DIMENSIONS
8BED              ;ONLY
8BED              ;
8BED  A9 0D       DUMP17 LDA #$0D        ;SEPARATE NORMAL
8BEF  20 D2 FF           JSR $FFD2      ; VARS FROM ARRAYS WITH
8BF2  A5 2F              LDA $2F        ; A CARRIAGE RETURN
8BF4  85 5F              STA $5F        ;SET POINTER TO 1ST
8BF6  A5 30              LDA $30        ; ARRAY
8BF8  85 60              STA $60
8BFA              ;
8BFA  A5 60       DUMP18 LDA $60         ;END OF ARRAYS?
8BFC  C5 32              CMP $32
8BFE  D0 06              BNE DUMP19     ;NO
8C00  A5 5F              LDA $5F
8C02  C5 31              CMP $31
8C04  F0 AD              BEQ DUMP09
8C06              ;
8C06  20 E1 FF    DUMP19 JSR $FFE1       ;STOP KEY?
8C09  F0 A8              BEQ DUMP09     ;YES,EXIT
8C0B  20 C5 8B           JSR DUMP12     ;GET AND PRINT NAME
8C0E  A5 25              LDA $25        ;WHICH TYPE?
8C10  F0 0A              BEQ DUMP21     ;REAL
8C12  C9 02              CMP #$02       ;STRING?
8C14  D0 03              BNE DUMP20     ;NO, ARRAY IS INTEGER
8C16  A9 24              LDA #$24       ;CHAR '$'
8C18  2C                 .BYT $2C       ;SKIP 2 BYTES
8C19  A9 25       DUMP20 LDA #$25        ;CHAR '%'
8C1B  2C                 .BYT $2C       ;SKIP 2 BYTES
8C1C  A9 20       DUMP21 LDA #$20        ;CHAR ' '
8C1E  20 D2 FF           JSR $FFD2      ;PRINT IT
8C21  20 E3 8B           JSR DUMP15
8C24  A9 20              LDA #$20
8C26  20 D2 FF           JSR $FFD2      ;ONE EXTRA SPACE
8C29  A9 28              LDA #$28       ;CHAR '('
8C2B  20 D2 FF           JSR $FFD2      ;PRINT IT
8C2E  A5 5F              LDA $5F
8C30  18                 CLC
8C31  69 03              ADC #$03       ;SET POINTER TO END
8C33  85 FB              STA $FB        ; OF ARRAY ENTRY FOR
8C35  A5 60              LDA $60        ; DISPLAY OF DIMS
```

```
LOC   CODE          LINE


8C37  69 00                ADC #$00
8C39  85 FC                STA $FC
8C3B  A0 01                LDY #$01
8C3D  B1 FB                LDA ($FB),Y    ;# OF DIMENSIONS
8C3F  85 FD                STA $FD
8C41  A9 00                LDA #$00
8C43  85 FE                STA $FE
8C45  06 FD                ASL $FD        ;TIMES 2
8C47  26 FE                ROL $FE
8C49  A5 FD                LDA $FD        ;PLUS END VALUE
8C4B  18                   CLC
8C4C  65 FB                ADC $FB
8C4E  85 FD                STA $FD
8C50  A5 FE                LDA $FE
8C52  65 FC                ADC $FC
8C54  85 FE                STA $FE
8C56                ;
8C56  A0 00         DUMP22 LDY #$00        ;GET DIMENSION VALUE
8C58  B1 FD                LDA ($FD),Y
8C5A  8D C9 8C             STA DIMENS+1
8C5D  C8                   INY
8C5E  B1 FD                LDA ($FD),Y
8C60  8D C8 8C             STA DIMENS
8C63  D0 03                BNE DUMP23     ;MINUS 1
8C65  CE C9 8C             DEC DIMENS+1
8C68  CE C8 8C      DUMP23 DEC DIMENS
8C6B  AD C9 8C             LDA DIMENS+1   ;PRINT NUMBER
8C6E  AE C8 8C             LDX DIMENS     ; IN .A(HI), .X(LO)
8C71  A4 5F                LDY $5F        ;SAVE ARRAY POINTER
8C73  8C C8 8C             STY DIMENS
8C76  A4 60                LDY $60
8C78  8C C9 8C             STY DIMENS+1
8C7B  20 CD BD             JSR $BDCD
8C7E  AC C8 8C             LDY DIMENS     ;RESTORE ARRAY POINTER
8C81  84 5F                STY $5F
8C83  AC C9 8C             LDY DIMENS+1
8C86  84 60                STY $60
8C88  38                   SEC            ;SUBTRACT 2 FROM
8C89  A5 FD                LDA $FD        ; DIMENSION POINTER
8C8B  E9 02                SBC #$02
8C8D  85 FD                STA $FD
8C8F  A5 FE                LDA $FE
8C91  E9 00                SBC #$00
8C93  85 FE                STA $FE
8C95  C5 FC                CMP $FC        ;END OF ARRAY?
8C97  D0 06                BNE DUMP24     ;NO
8C99  A5 FD                LDA $FD
8C9B  C5 FB                CMP $FB
8C9D  F0 08                BEQ DUMP25     ;YES
8C9F                ;
8C9F  A9 2C         DUMP24 LDA #$2C        ;CHAR ','
8CA1  20 D2 FF             JSR $FFD2      ;PRINT IT
8CA4  4C 56 8C             JMP DUMP22     ;DO NEXT ELEMENT
8CA7                ;
8CA7  A0 03         DUMP25 LDY #$03       ;GET LENGTH OF
8CA9  B1 5F                LDA ($5F),Y    ;ARRAY ENTRY
8CAB  85 FB                STA $FB
8CAD  88                   DEY
8CAE  B1 5F                LDA ($5F),Y
8CB0  18                   CLC
8CB1  65 5F                ADC $5F        ;AND ADD TO ARRAY
8CB3  85 5F                STA $5F        ; POINTER
8CB5  A5 60                LDA $60
8CB7  65 FB                ADC $FB
8CB9  85 60                STA $60
```

```
LOC   CODE          LINE


8CBB  A9 29                LDA #$29        ;CHAR ')'
8CBD  20 D2 FF             JSR $FFD2       ;PRINT IT
8CC0  A9 0D                LDA #$0D        ;CARRIAGE RETURN
8CC2  20 D2 FF             JSR $FFD2       ;PRINT IT
8CC5  4C FA 8B             JMP DUMP18      ;DO NEXT ARRAY
8CC8  00 00         DIMENS .WOR 0
8CCA  22            DUMTBL .BYT $22,$20,$3D,$24
8CCB  20
8CCC  3D
8CCD  24
8CCE                       .END
```

EXEC

Abbreviated entry: E(shift)X

Affected Basic abbreviations: EXP - EXP

Token: Hex $EE,$0D Decimal 238,13

Modes: Direct and program

Recommended mode: Direct only

Purpose: To EXECute a text file stored on disk. This command works in conjunction with GET and PUT.

Syntax: EXEC filename,d - where d is the device number (disk only).

Errors: Illegal device - if the device number specified is less than eight
Missing filename - if a null filename is specified
File not found - if the file does not exit
Device not present - if no disk drive is connected
Too many files - if ten files are already open
Disk errors - at the end, the disk error channel is read and displayed

Use: EXEC can be used in several different ways. The main one is to set up function keys when first powered up. For example, enter the program:

```
 10 CTL(,,5,0,0,1)
 20 KEY1,"CATALOG"+CHR$(13)
 30 KEY2,"DISK"+CHR$(13)
 40 KEY3,"LIST"+CHR$(13)
 50 KEY4,"RUN"+CHR$(13)
 60 KEY5,"OLD"+CHR$(13)
 70 KEY6,"PEEK("
 80 KEY7,"RENUMBER"
 90 KEY8,"FIND@"
100 PRINT CTL(12,12,,,,1)"FUNCTION KEYS DEFINED"
```

Use the PUT command to write this to a disk file: PUT“FK”,8

When powered up, type EXEC“FK”,8 and the commands will be carried out and your function keys will be defined.

Other uses could be a string of CHANGE commands to a program.

Routine entry point: $8CCE

Routine operation: The filename and device number are read in and the file is opened. Each line is read into the input buffer until carriage return is found. It is then tokenised, and executed until the file is complete or an operating error occurs.

```
LOC    CODE          LINE


8CCE                        .LIB EXEC
8CCE   20 6F 98      EXEC   JSR DPARS        ;GET FILE PARAMETERS
8CD1   20 B7 8F             JSR GETOPN       ;OPEN FILE
8CD4   A9 93                LDA #$93         ;CLEAR SCREEN
8CD6   20 D2 FF             JSR $FFD2
8CD9   AD 00 03             LDA $0300        ;STORE OFF ERROR LINK
8CDC   8D 90 8D             STA EXECER
8CDF   AD 01 03             LDA $0301
8CE2   8D 91 8D             STA EXECER+1
8CE5   AD 02 03             LDA $0302        ;STORE OFF WARM START
8CE8   8D 8E 8D             STA EXECST
8CEB   AD 03 03             LDA $0303
8CEE   8D 8F 8D             STA EXECST+1
8CF1   A9 60                LDA #<MERGRT     ;SET 'RESET INPUT'
8CF3   8D 2C 03             STA $032C        ; TO RTS
8CF6   A9 98                LDA #>MERGRT
8CF8   8D 2D 03             STA $032D
8CFB   A9 59                LDA #<EXEC06     ;SET ERROR VECTOR
8CFD   8D 00 03             STA $0300
8D00   A9 8D                LDA #>EXEC06
8D02   8D 01 03             STA $0301
8D05   A9 0F                LDA #<EXEC02     ;SET WARM START
8D07   8D 02 03             STA $0302
8D0A   A9 8D                LDA #>EXEC02
8D0C   8D 03 03             STA $0303
8D0F   AE 92 8D      EXEC02 LDX EXECNO
8D12   20 C6 FF             JSR $FFC6        ;SET INPUT
8D15   A2 18                LDX #24          ;BOTTOM
8D17   A0 00                LDY #$00         ; LEFT
8D19   18                   CLC
8D1A   20 F0 FF             JSR $FFF0        ; OF SCREEN
8D1D   A2 00                LDX #$00
8D1F   20 CF FF      EXEC03 JSR $FFCF        ;GET BYTE
8D22   48                   PHA
8D23   A5 90                LDA $90          ;CHECK STATUS
8D25   D0 29                BNE EXEC05
8D27   68                   PLA
8D28   C9 0D                CMP #$0D         ;CARRIAGE RETURN?
8D2A   F0 0A                BEQ EXEC04
8D2C   9D 00 02             STA $0200,X
8D2F   E8                   INX
8D30   20 D2 FF             JSR $FFD2        ;PRINT CHAR
8D33   4C 1F 8D             JMP EXEC03
8D36   A9 00         EXEC04 LDA #$00
8D38   9D 00 02             STA $0200,X
8D3B   A9 01                LDA #$01
8D3D   85 C6                STA $C6
8D3F   A9 0D                LDA #$0D
8D41   20 D2 FF             JSR $FFD2
8D44   A2 00                LDX #$00         ;SET KEYBOARD AS INPUT
```

```
LOC    CODE         LINE


8D46   20 C6 FF             JSR $FFC6
8D49   A2 FF                LDX #$FF
8D4B   A0 01                LDY #$01
8D4D   4C 86 A4             JMP $A486        ;EXEC IT
8D50   20 65 8D     EXEC05  JSR EXEC07        ;RESET VECTORS
8D53   20 55 8A             JSR DISK01       ;DISPLAY DISK ERROR
8D56   4C 74 A4             JMP $A474        ;EXIT TO READY
8D59   90 B4        EXEC06  BCC EXEC02
8D5B   8A                   TXA              ;SAVE ERROR NUMBER
8D5C   48                   PHA
8D5D   20 65 8D             JSR EXEC07       ;RESET VECTORS
8D60   68                   PLA              ;RESTORE ERROR NUMBER
8D61   AA                   TAX
8D62   6C 00 03             JMP ($0300)      ;SEND ERROR
8D65   A9 2F        EXEC07  LDA #$2F          ;RESTORE 'RESET DEFAULT IO'
8D67   8D 2C 03             STA $032C
8D6A   A9 F3                LDA #$F3
8D6C   8D 2D 03             STA $032D
8D6F   AD 90 8D             LDA EXECER       ;RESET ERROR LINK
8D72   8D 00 03             STA $0300
8D75   AD 91 8D             LDA EXECER+1
8D78   8D 01 03             STA $0301
8D7B   AD 8E 8D             LDA EXECST       ;RESET WARM START
8D7E   8D 02 03             STA $0302
8D81   AD 8F 8D             LDA EXECST+1
8D84   8D 03 03             STA $0303
8D87   AE 92 8D             LDX EXECNO
8D8A   20 C3 FF             JSR $FFC3        ;CLOSE FILE
8D8D   60                   RTS
8D8E   00 00        EXECST  .WOR 0
8D90   00 00        EXECER  .WOR 0
8D92   00           EXECNO  .BYT 0
8D93                        .END
```

FIND

Abbreviated entry: F(shift)I

Affected Basic abbreviations: None

Token: Hex $EE,$0E Decimal 238,14

Modes: Direct and program

Recommended mode: Direct only

Purpose: To find all occurrences of a string or command inside a Basic program.

Syntax: FIND string – where d is the delimiter character as in CHANGE.

Errors: Syntax error – if the syntax is not as above
String too long – if the string is longer than 40 characters

Use: FIND is another useful routine for de-bugging and checking Basic programs. An example of FIND is:

FIND @PRINT@

which will find and list all lines containing the command PRINT. If PRINT occurs more than once on a line, the line will be listed each time it is found with the exception of the last lines, where the line will be listed only once.

Routine entry point: $8D93

Routine operation: The string to be found is read in within quotes, including spaces and colons, and stored away. The rest of the program is a loop that searches the program until the string has been found, lists the line, and starts searching from the next character.

The error message vector is stored away and replaced with a jump to an 'RTS' so that LIST will return to the routine.

```
LOC   CODE        LINE


8D93                      .LIB FIND
8D93  20 91 8E    FIND    JSR FIND14      ;GET CHARACTER
8D96  85 59               STA $59        ;STORE IN FLAG
8D98  A2 00               LDX #$00
8D9A  20 C7 8D            JSR FIND03     ;GET SEARCH STRING
8D9D  20 E5 8D            JSR FIND05     ;SETUP POINTERS
8DA0  78                  SEI
8DA1  AD 00 03            LDA $0300
8DA4  8D CF 8E            STA FINDER
8DA7  AD 01 03            LDA $0301
8DAA  8D D0 8E            STA FINDER+1
8DAD  A9 67               LDA #<FIND11   ;ERROR LINK TO RTS
8DAF  8D 00 03            STA $0300
8DB2  A9 8E               LDA #>FIND11
8DB4  8D 01 03            STA $0301
8DB7  58                  CLI
8DB8  20 F3 8D            JSR FIND06     ;FIND STRING
8DBB  20 68 8E    FIND01  JSR FIND12      ;LIST LINE
8DBE  20 F9 8D            JSR FIND07     ;FIND STRING
8DC1  4C BB 8D            JMP FIND01     ;AND REPEAT
8DC4              ;
8DC4  4C 08 AF    FIND02  JMP $AF08       ;SEND SYNTAX ERROR
8DC7              ;
8DC7  20 8B 8E    FIND03  JSR FIND13      ;GET A CHARACTER
8DCA  F0 F8               BEQ FIND02     ;END OF LINE
8DCC  C5 59               CMP $59        ;END OF STRING?
8DCE  F0 0D               BEQ FIND04     ;YES, COMPLETE
8DD0  9D 40 BF            STA $BF40,X    ;STORE IN SEARCH STRING
8DD3  E8                  INX
8DD4  E0 40               CPX #$40       ;STRING TOO LONG?
8DD6  D0 EF               BNE FIND03     ;NO
8DD8  A2 17               LDX #$17       ;STRING TOO LONG
8DDA  4C 37 A4            JMP $A437      ;OUTPUT ERROR
8DDD  A9 00       FIND04  LDA #$00        ;TERMINATOR TO STRING
8DDF  9D 40 BF            STA $BF40,X    ;STORE IT
8DE2  86 22               STX $22        ;STORE STRING LENGTH
8DE4  60                  RTS            ;EXIT
8DE5              ;
8DE5  A5 2B       FIND05  LDA $2B         ;GET START OF PROGRAM
8DE7  18                  CLC
8DE8  69 02               ADC #$02       ;PLUS 2
8DEA  85 57               STA $57
8DEC  A5 2C               LDA $2C        ;GET START OF PROG MSB
8DEE  69 00               ADC #$00
8DF0  85 58               STA $58        ;STORE IT
```

```
LOC   CODE        LINE


8DF2  60                 RTS
8DF3              ;
8DF3  A2 00       FIND06 LDX #$00        ;INDEX TO STRING
8DF5  A0 02              LDY #$02       ;INDEX TO LINE
8DF7  84 23              STY $23
8DF9  A5 01       FIND07 LDA $01
8DFB  29 FE              AND #$FE       ;OUT BASIC ROM
8DFD  85 01              STA $01
8DFF  B1 57              LDA ($57),Y    ;GET BYTE
8E01  F0 21              BEQ FIND09     ;END OF LINE
8E03  DD 40 BF           CMP $BF40,X    ;SAME AS STRING?
8E06  08                 PHP
8E07  A5 01              LDA $01
8E09  09 01              ORA #$01       ;IN BASIC ROM
8E0B  85 01              STA $01
8E0D  28                 PLP
8E0E  D0 07              BNE FIND08     ;NOT MATCHED
8E10  C8                 INY            ;NEXT BYTE
8E11  E8                 INX            ;NEXT CHAR
8E12  E4 22              CPX $22        ;STRING MATCHED?
8E14  D0 E3              BNE FIND07     ;NO
8E16  60                 RTS            ;YES
8E17  E6 23       FIND08 INC $23         ;START AT NEXT BYTE
8E19  A4 23              LDY $23
8E1B  A2 00              LDX #$00       ;AND START OF STRING
8E1D  B1 57              LDA ($57),Y    ;GET BYTE
8E1F  F0 03              BEQ FIND09     ;END OF LINE
8E21  4C F9 8D           JMP FIND07     ;TRY AGAIN
8E24  A5 01       FIND09 LDA $01
8E26  09 01              ORA #$01       ;IN BASIC ROM
8E28  85 01              STA $01
8E2A  A5 57              LDA $57
8E2C  38                 SEC
8E2D  E9 02              SBC #$02       ;LINE POINTER -2
8E2F  85 57              STA $57
8E31  A5 58              LDA $58
8E33  E9 00              SBC #$00
8E35  85 58              STA $58
8E37  A0 00              LDY #$00
8E39  B1 57              LDA ($57),Y    ;GET LINK LO
8E3B  85 59              STA $59        ;STORE IT
8E3D  C8                 INY
8E3E  B1 57              LDA ($57),Y    ;GET LINK HI
8E40  85 58              STA $58        ;STORE TO POINTER HI
8E42  05 59              ORA $59        ;END OF PROGRAM?
8E44  F0 10              BEQ FIND10     ;YES
8E46  A5 59              LDA $59        ;GET LINE POINTER LO
8E48  18                 CLC
8E49  69 02              ADC #$02       ;ADD 2
8E4B  85 57              STA $57        ;STORE IT
8E4D  A5 58              LDA $58        ;GET HI BYTE
8E4F  69 00              ADC #$00
8E51  85 58              STA $58
8E53  4C F3 8D           JMP FIND06     ;DO NEXT LINE
8E56  78          FIND10 SEI
8E57  AD CF 8E           LDA FINDER     ;RESET ERROR LINK
8E5A  8D 00 03           STA $0300
8E5D  AD D0 8E           LDA FINDER+1
8E60  8D 01 03           STA $0301
8E63  58                 CLI
8E64  4C 74 A4           JMP $A474      ;EXIT
8E67              ;
8E67  60          FIND11 RTS             ;ERROR LINK
8E68              ;
8E68  A0 00       FIND12 LDY #$00
```

```
LOC    CODE        LINE

8E6A   20 96 8E           JSR FIND15     ;SAVE POINTERS
8E6D   A9 91              LDA #$91       ;CURSOR UP
8E6F   20 D2 FF           JSR $FFD2      ;PRINT IT
8E72   B1 57              LDA ($57),Y    ;GET LINE# LO
8E74   85 14              STA $14        ;STORE IT
8E76   C8                 INY
8E77   B1 57              LDA ($57),Y    ;GET LINE# HI
8E79   85 15              STA $15        ;STORE IT
8E7B   20 13 A6           JSR $A613      ;FIND LINE ADDRESS
8E7E   20 C9 A6           JSR $A6C9      ;LIST LINE
8E81   20 B0 8E           JSR FIND16     ;RESTORE POINTERS
8E84   E6 23              INC $23        ;NEXT CHAR IN LINE
8E86   A4 23              LDY $23
8E88   A2 00              LDX #$00       ;START OF STRING
8E8A   60                 RTS
8E8B               ;
8E8B   E6 7A       FIND13 INC $7A         ;INCREASE LSB
8E8D   D0 02              BNE FIND14
8E8F   E6 7B              INC $7B
8E91   A0 00       FIND14 LDY #$00
8E93   B1 7A              LDA ($7A),Y    ;GET INPUT BYTE
8E95   60                 RTS
8E96               ;
8E96   A5 22       FIND15 LDA $22         ;STORE STRING LENGTH
8E98   8D CA 8E           STA FIND17
8E9B   A5 23              LDA $23        ;STORE LINE INDEX
8E9D   8D CB 8E           STA FIND17+1
8EA0   A5 57              LDA $57        ;STORE LINE POINTER LO
8EA2   8D CC 8E           STA FIND17+2
8EA5   A5 58              LDA $58        ;HI
8EA7   8D CD 8E           STA FIND17+3
8EAA   A5 FC              LDA $FC        ;SAVE CHANGE VARIABLE
8EAC   8D CE 8E           STA FIND17+4
8EAF   60                 RTS
8EB0               ;
8EB0   AD CA 8E    FIND16 LDA FIND17      ;GET STRING LENGTH
8EB3   85 22              STA $22
8EB5   AD CB 8E           LDA FIND17+1   ;GET LINE INDEX
8EB8   85 23              STA $23
8EBA   AD CC 8E           LDA FIND17+2   ;GET LINE POINTER LO
8EBD   85 57              STA $57
8EBF   AD CD 8E           LDA FIND17+3   ;GET LINE POINTER HI
8EC2   85 58              STA $58
8EC4   AD CE 8E           LDA FIND17+4   ;GET CHANGE PARAMETER
8EC7   85 FC              STA $FC
8EC9   60                 RTS
8ECA   00          FIND17 .BYT $00,$00,$00,$00,$00
8ECB   00
8ECC   00
8ECD   00
8ECE   00
8ECF   00 00       FINDER .WOR 0
8ED1                      .END
```

GET

Abbreviated entry: G(shift)E

Affected Basic abbreviations: None

Token: Hex $EE,$0F Decimal 238,15

Modes: Direct and program

Recommended mode: Either; different effects in direct mode and program mode.

Purpose: To input an ASCII file on disk into memory with line numbers created from 1000 in steps of 10. GET will read in files created by the Commodore assembler and SYSRES. Each line is read in until a carriage return is reached. It is then tokenised and entered into memory as a program line.

Syntax: Direct mode: GET filename, d – where d is the device number (disk only)
Run mode : as chapter 3 GET and GET#

Errors: Illegal device – if the device number specified is less than eight
Missing file name – if a null filename is specified
File not found – if the file does not exist
Device not present – if no disk drive is connected
File open error – if ten files are already open
Disk errors – at the end, the disk error channel is read and displayed

Use: For editing Commodore assembler files or for editing files for the use of the EXEC command.

Routine entry point: $8ED1

Routine operation: The GET routine first checks whether the computer is in run mode or direct. If it is in run mode, then the Basic version of GET is performed. If in direct mode, the file parameters are read in and checked for a null filename or the device not being disk. If these checks are OK, the message 'reading' filename is displayed and the file is opened. Each line is then input and stored in the input buffer, tokenised, and entered into memory until the end of file marker is reached. The program is then re-chained and the variable pointers are set to the correct values for the program. Finally the disk error channel is read and displayed.

```
LOC   CODE         LINE


8ED1                       .LIB GET
8ED1  A5 9D        GET     LDA $9D          ;CHECK IF DIRECT
8ED3  D0 06                BNE GETUN        ;YES, DIRECT
8ED5  20 79 00             JSR $0079        ;GET CURRENT CHAR
8ED8  4C 7E AB             JMP $AB7E        ;PERFORM BASIC 'GET'
8EDB  20 6F 98     GETUN   JSR DPARS        ;GET FILE PARAMETERS
8EDE  20 99 8F             JSR GETMES       ;'READING'
8EE1  20 B7 8F             JSR GETOPN       ;OPEN FILE
8EE4  20 AC 8F             JSR GETIN        ;SET INPUT
8EE7  A5 2B                LDA $2B          ;SET START OF PROGRAM
8EE9  85 FB                STA $FB          ;POINTER
8EEB  A5 2C                LDA $2C
8EED  85 FC                STA $FC
```

```
LOC   CODE        LINE

8EEF  A5 2B              LDA $2B
8EF1  18                 CLC
8EF2  69 02              ADC #$02
8EF4  AA                 TAX
8EF5  A5 2C              LDA $2C
8EF7  69 00              ADC #$00
8EF9  85 2E              STA $2E
8EFB  85 30              STA $30
8EFD  85 32              STA $32
8EFF  86 2D              STX $2D
8F01  86 2F              STX $2F
8F03  86 31              STX $31
8F05  A9 03              LDA #$03       ;START LINE# HI
8F07  A2 E8              LDX #$E8       ;START LINE# HI
8F09  8D DD 8F           STA GETLNO+1
8F0C  8E DC 8F           STX GETLNO
8F0F  A0 00       GETLP1 LDY #$00
8F11  20 CF FF    GETLP2 JSR $FFCF       ;INPUT BYTE
8F14  C9 0D              CMP #$0D       ;END OF LINE?
8F16  F0 0C              BEQ GETLN      ;YES
8F18  C9 0A              CMP #$0A       ;LINE FEED?
8F1A  F0 F5              BEQ GETLP2     ;YES
8F1C  99 00 02           STA $0200,Y    ;STORE BYTE
8F1F  C8                 INY
8F20  C0 57              CPY #$57       ;END OF BUFFER?
8F22  D0 ED              BNE GETLP2
8F24  A5 90       GETLN  LDA $90         ;STATUS
8F26  8D DE 8F           STA GETER
8F29  A9 00              LDA #$00       ;TERMINATOR
8F2B  99 00 02           STA $0200,Y    ;STORE
8F2E  A2 00              LDX #$00
8F30  86 7A              STX $7A
8F32  A9 02              LDA #$02
8F34  85 7B              STA $7B
8F36  20 79 A5           JSR $A579      ;CRUNCH LINE
8F39  AD 00 02           LDA $0200
8F3C  F0 42              BEQ GETLP4     ;NULL LINE
8F3E  A0 02              LDY #$02
8F40  AD DC 8F           LDA GETLNO     ;LINE# LO
8F43  91 FB              STA ($FB),Y    ;STORE IT
8F45  C8                 INY
8F46  AD DD 8F           LDA GETLNO+1   ;LINE# HI
8F49  91 FB              STA ($FB),Y    ;STORE IT
8F4B  C8          GETLP3 INY
8F4C  B9 FC 01           LDA $01FC,Y    ;GET BYTE
8F4F  91 FB              STA ($FB),Y    ;STORE IT
8F51  D0 F8              BNE GETLP3     ;UNTIL END OF LINE
8F53  C8                 INY
8F54  98                 TYA
8F55  A0 00              LDY #$00
8F57  18                 CLC
8F58  65 FB              ADC $FB        ;INCREASE POINTER BY
8F5A  85 FD              STA $FD        ;LENGTH
8F5C  91 FB              STA ($FB),Y
8F5E  A5 FC              LDA $FC
8F60  69 00              ADC #$00
8F62  C8                 INY
8F63  91 FB              STA ($FB),Y
8F65  A8                 TAY
8F66  A5 FD              LDA $FD
8F68  85 FB              STA $FB
8F6A  84 FC              STY $FC
8F6C  98                 TYA
8F6D  30 19              BMI GETEND
8F6F  AD DC 8F           LDA GETLNO     ;INCREASE LINE#
8F72  18                 CLC
```

```
LOC   CODE         LINE


8F73  69 0A                ADC #$0A       ;BY 10
8F75  8D DC 8F             STA GETLNO
8F78  AD DD 8F             LDA GETLNO+1
8F7B  69 00                ADC #$00
8F7D  8D DD 8F             STA GETLNO+1
8F80  AD DE 8F     GETLP4  LDA GETER       ;STATUS?
8F83  D0 03                BNE GETEND     ;BAD
8F85  4C 0F 8F             JMP GETLP1     ;DO NEXT LINE
8F88               ;
8F88  A9 00        GETEND  LDA #$00
8F8A  A8                   TAY
8F8B  91 FB                STA ($FB),Y    ;ZERO END OF PROGRAM
8F8D  C8                   INY
8F8E  91 FB                STA ($FB),Y
8F90  20 AC 99             JSR PUTEND     ;CLOSE AND DISK
8F93  20 85 98             JSR OLD        ;RESET POINTERS
8F96  4C 74 A4             JMP $A474
8F99  A9 A3        GETMES  LDA #<GMESSG    ;POINTER TO
8F9B  A0 8F                LDY #>GMESSG   ;'READING'
8F9D  20 1E AB             JSR $AB1E      ;PRINT STRING
8FA0  4C C1 F5             JMP $F5C1      ;PRINT FILENAME
8FA3  52 45        GMESSG  .BYT 'READING ',$00
8FAB  00
8FAC  A6 B8        GETIN   LDX $B8
8FAE  20 C6 FF             JSR $FFC6      ;SET INPUT
8FB1  B0 01                BCS GETIN1     ;ERROR
8FB3  60                   RTS
8FB4  4C F9 E0     GETIN1  JMP $E0F9       ;SEND ERROR
8FB7               ;
8FB7  A0 00        GETOPN  LDY #$00
8FB9  B1 BB        GETOP1  LDA ($BB),Y     ;GET BYTE
8FBB  99 00 02             STA $0200,Y    ;STORE IT
8FBE  C8                   INY
8FBF  C4 B7                CPY $B7        ;END OF FILENAME?
8FC1  D0 F6                BNE GETOP1     ;NOT YET
8FC3  A2 00                LDX #$00
8FC5  BD D8 8F     GETOP2  LDA GETSR,X     ;GET BYTE
8FC8  99 00 02             STA $0200,Y    ;STORE IT
8FCB  E8                   INX
8FCC  C8                   INY
8FCD  E0 04                CPX #$04       ;END OF SR?
8FCF  D0 F4                BNE GETOP2     ;NOT YET
8FD1  A9 60                LDA #$60
8FD3  85 B9                STA $B9
8FD5  4C 16 9A             JMP PUTOP4     ;COMPLETE OPEN
8FD8  2C 53        GETSR   .BYT ',S,R'
8FDC  00 00        GETLNO  .WOR 0
8FDE  00           GETER   .BYT 0
8FDF                       .END
```

HIMEM

Abbreviated entry: H(shift)I

Affected Basic abbreviations: None

Token: Hex $EE,$1E   Decimal 238,30

Modes: Direct and program

Recommended mode: Either

Purpose: To read/set the top of Basic programming memory.

Syntax: HIMEM = expression - sets the top of memory to the expression (0-65535)
A = HIMEM or PRINT HIMEM - returns the top of memory address

Errors: Syntax error
Illegal quantity - if the address is out of the range (0-65535)

Use: HIMEM can be used to protect an area of memory at the top of Basic programming memory for the use of data storage or machine code programs. With these routines in memory, HIMEM is set at 32768. When HIMEM is used to set, a CLR is performed, thus wiping out all variables.

Routine entry point: $8FDF

Routine operation: HIMEM first checks to see whether it was called by the arithmetic routine or the execute statement routine. If the arithmetic routine called it, the top of memory pointer is read and converted to floating point form. If not, the '=' sign is scanned and the value is read in and stored at the top of memory pointer. CLR is then performed.

```
LOC   CODE        LINE


8FDF                     .LIB HIMEM
8FDF  68          HIMEM  PLA              ;GET RETURN ADDRESS
8FE0  48                 PHA
8FE1  C9 8C              CMP #$8C        ;ARITHMETIC?
8FE3  D0 07              BNE HIMSET      ;NO
8FE5  A6 37              LDX $37         ;GET HIMEM LO
8FE7  A5 38              LDA $38         ;GET HIMEM HI
8FE9  4C A3 89           JMP ASSIGN      ;SEND IT
8FEC              ;
8FEC  A9 B2       HIMSET LDA #$B2         ;CHAR '='
8FEE  20 FF AE           JSR $AEFF       ;SCAN PAST '='
8FF1  20 8A AD           JSR $AD8A       ;GET ADDRESS
8FF4  20 F7 B7           JSR $B7F7       ;FIX IT
8FF7  A5 14              LDA $14         ;GET VALUE LO
8FF9  85 37              STA $37         ;STORE TO MEMTOP
8FFB  85 35              STA $35         ;UTILITY STRING
8FFD  85 33              STA $33         ;STRING
8FFF  A5 15              LDA $15         ;GET VALUE HI
9001  85 38              STA $38         ;STORE TO HI BYTES
9003  85 36              STA $36
9005  85 34              STA $34
9007  A5 2D              LDA $2D         ;PERFORM CLR
9009  85 2F              STA $2F
900B  85 31              STA $31
900D  A5 2E              LDA $2E
900F  85 30              STA $30
9011  85 32              STA $32
9013  60                 RTS
9014                     .END
```

KEY

Abbreviated entry: K(shift)E

Affected Basic abbreviations: None

Token: Hex $EE,$10 Decimal 238,16

Modes: Direct and program

Recommended mode: Either, but function keys work in direct mode only.

Purpose: To set an eight byte string to one of the eight function keys.

Syntax: KEY expression, string – where the expression is a value (1–8) and the string is any string expression (first eight bytes only are accepted).

Errors: Illegal quantity – if the key number is <1 or >8
Syntax error – if missing comma
String too long – if the string is longer than 255 bytes
Type mismatch – if the command is numeric instead of string

Use: KEY is used to set a commonly used string or command onto a function key. There are eight function keys available and each one can be eight bytes long. For an example of the format for KEY, see the EXEC command.

Routine entry point: $9014

Routine operation: KEY first reads in the function key number and checks that it is within range (anything after a decimal point is ignored). If it is within range, the comma is scanned past and the string is read in. The string is then copied into the storage area until the whole string is in or the first eight bytes.

```
LOC   CODE       LINE


9014                    .LIB KEY
9014  20 9E B7   KEY    JSR $B79E        ;GET KEY#
9017  E0 00             CPX #$00        ;IN RANGE?
9019  F0 04             BEQ KEYERR      ;NO
901B  E0 09             CPX #$09
901D  90 05             BCC KEY01       ;YES
901F  A2 0E      KEYERR LDX #$0E         ;ILLEGAL QUANTITY
9021  4C 37 A4          JMP $A437       ;SEND ERROR
9024  A9 BF      KEY01  LDA #$BF         ;POINTER HI BYTE
9026  85 FC             STA $FC
9028  CA                DEX
9029  BD 54 90          LDA KEYLO,X     ;GET LO BYTE
902C  85 FB             STA $FB
902E  20 FD AE          JSR $AEFD       ;SCAN PAST COMMA
9031  20 9E AD          JSR $AD9E       ;GET STRING
9034  8D 5C 90          STA STLEN
9037  20 A3 B6          JSR $B6A3       ;DISCARD STRING
903A  A0 00             LDY #$00
903C  B1 22      KEY02  LDA ($22),Y      ;GET BYTE
903E  91 FB             STA ($FB),Y     ;STORE IT
9040  C8                INY
```

```
LOC   CODE          LINE


9041  CC 5C 90             CPY STLEN        ;END OF STRING?
9044  F0 05                BEQ KEY04
9046  C0 08                CPY #$08         ;END OF ROOM?
9048  D0 F2                BNE KEY02        ;NOT YET
904A  60            KEY03  RTS
904B  C0 08         KEY04  CPY #$08          ;STRING LENGTH=8?
904D  F0 FB                BEQ KEY03        ;YES
904F  A9 00                LDA #$00         ;ZERO TERMINATOR
9051  91 FB                STA ($FB),Y      ;STORE IT
9053  60                   RTS
9054  C0            KEYLO  .BYT $C0,$E0,$C8,$E8
9055  E0
9056  C8
9057  E8
9058  D0                   .BYT $D0,$F0,$D8,$F8
9059  F0
905A  D8
905B  F8
905C  00            STLEN  .BYT 0
905D                       .END
```

LOMEM

Abbreviated entry: L(shift)O

Affected Basic abbreviations: LOAD – LO(shift)A

Token: Hex $EE,$1F Decimal 238,31

Modes: Direct and program

Recommended mode: Either

Purpose: To read/set the bottom of Basic programming memory.

Syntax: LOMEM = expression – sets the bottom of memory to the expression (0–65535).
A = LOMEM or PRINT LOMEM – returns the bottom of memory address.

Errors: Syntax error
Illegal quantity – if the address is out of range (0–65535)

Use: LOMEM can be used to protect an area of memory at the bottom of the Basic programming memory for the use of data storage or machine code programs. LOMEM is originally set at 2049. When LOMEM is used to set, a NEW is performed, thus wiping out all variables and Basic program at the new address. If a program was there, use OLD to restore it.

Routine entry point: $905D

Routine operation: LOMEM first checks to see whether it was called by the arithmetic routine or the execute statement routine. If the arithmetic routine called it, the bottom of memory pointer is read and converted to floating point form. If not, the '=' sign is scanned and the value is read in and stored at the bottom of memory pointer. NEW is then performed. The byte below the new bottom of memory is also set to zero.

```
LOC   CODE        LINE


905D                     .LIB LOMEM
905D  68          LOMEM  PLA              ;GET RETURN ADDRESS
905E  48                 PHA
905F  C9 8C              CMP #$8C        ;ARITHMETIC?
9061  D0 07              BNE LOMSET      ;NO
9063  A6 2B              LDX $2B         ;GET LOMEM LO
9065  A5 2C              LDA $2C         ;HI
9067  4C A3 89           JMP ASSIGN      ;SEND IT
906A              ;
906A  A9 B2       LOMSET LDA #$B2         ;TOKEN '='
906C  20 FF AE           JSR $AEFF       ;SCAN FAST '='
906F  20 8A AD           JSR $AD8A       ;GET ADDRESS
9072  20 F7 B7           JSR $B7F7       ;FIX IT
9075  A5 14              LDA $14         ;GET LO BYTE
9077  85 2B              STA $2B         ;STORE BOTTOM
9079  18                 CLC
907A  69 02              ADC #$02        ;SET UP VARS
907C  85 2D              STA $2D
907E  85 2F              STA $2F
9080  85 31              STA $31
9082  A5 15              LDA $15         ;GET HI BYTE
9084  85 2C              STA $2C         ;STORE BOTTOM
9086  69 00              ADC #$00
9088  85 2E              STA $2E         ;SET UP VARS
908A  85 30              STA $30
908C  85 32              STA $32
908E  A5 14              LDA $14         ;COMPLETE NEW
9090  D0 02              BNE LOM01
9092  C6 15              DEC $15
9094  C6 14       LOM01  DEC $14
9096  A0 02              LDY #$02        ;LOOP TO STORE 3 ZEROS
9098  A9 00              LDA #$00
909A  91 14       LOM02  STA ($14),Y      ;STORE ZERO
909C  88                 DEY
909D  10 FB              BPL LOM02       ;AND NEXT
909F  A5 38              LDA $38         ;RESET STRING POINTERS
90A1  85 36              STA $36
90A3  85 34              STA $34
90A5  A5 37              LDA $37
90A7  85 35              STA $35
90A9  85 33              STA $33
90AB  60                 RTS
90AC                     .END
```

MAT

Abbreviated entry: M(shift)A

Token: Hex $EE,$11   Decimal 238,17

Modes: Program and direct

Purpose: To perform arithmetic operations on entire arrays, assuming their contents to be matrices.

Syntax: MAT array name = (arithmetic expression). Assign scalar value to all elements of the matrix in the array. Brackets are required around the expression.

MAT array name = array name. Assign all corresponding elements from one array to another. Both arrays must be numeric and of the same dimensions.

MAT array name = array name operator (arithmetic expression) or
MAT array name = (arithmetic expression) operator array name. The operator may be + or * to add or multiply a matrix with a scalar value.

MAT array name = array name + array name. All three arrays must be of the same dimensions and numeric.

MAT array name = array name * array name. Array sizes must follow the convention for matrix multiplication i.e. (a × c) = (a × b)*(b × c), where a,b,c are the array sizes in the DIM statement plus 1 (element 0 is used).

The MAT command will only accept arrays of 1 or 2 dimensions, of only numeric type and with not more than 255 elements in either dimension.

Errors: Syntax error - when the expression is not in brackets or an illegal operator is used
Type mismatch - for string arrays
Bad subscript - for arrays of incorrect size etc.

Use: High speed matrix arithmetic is approximately eight times faster than an equivalent basic subroutine. Using this command also saves the use of nested FOR...NEXT loops, thereby reducing the chances of an Out of memory error due to the stack being full. Since most versions of Basic on mainframe computers have full matrix arithmetic, this subset of the full MAT command will be useful in converting programs to run on the CBM 64. Matrix arithmetic is often used in programs handling large amounts of numbers in linear equations.

The routine uses the simple convention that a matrix of size a × b will be stored in an array dimensioned by DIM A(a−1,b−1). This means that a routine to read a 5 × 2 matrix from data statements would be:

```
DIM A(4,1)

FOR I = 0 TO 4
FOR J = 0 TO 1
READ A(I,J)
NEXT J,I

DATA 0,4
DATA 3 , 5
DATA -5,3.45
DATA 1 , 1
DATA .4,-4
```

To print an array use a routine like:

```
FOR I = 0 TO 4
FOR J = 0 TO 1
PRINT A(I,J),
NEXT J
PRINT
NEXT I
```

The matrix multiplication is equivalent to: (a × c) = (a × b) * (b × c).

```
DIM A(a-1,c-1),B(a-1,b-1)c,(b-1,c-1)
MAT A = B * C
```

is the same as but faster than:

```
FOR I = 0 TO a-1
FOR J = 0 TO c-1
T = 0
FOR K = 0 TO b-1
T = T + B(J,K) * C(K,I)
NEXT K
A(J,I) = T
NEXT J
NEXT I
```

Routine entry point: $90AC

Routine operation: The MAT routine uses the following Basic ROM calls:

$AEF1 – Evaluate expression in brackets
$BBD4 – FAC#1 to memory (x.y)
$BBA2 – Memory (x.y) to FAC#1
$B1BF – Float to fixed
$B391 – Fixed to float
$B867 – Memory (a.y) + FAC#1 to FAC#1
$B850 – Memory (a.y) − FAC#1 to FAC#1
$BA28 – Memory (a.y) * FAC#1 to FAC#1

The routine for assignment will, for speed, perform just a block memory move if the two arrays are both of the same type e.g. both integer. The multiply routine works in the same way as the Basic version above. It calculates the address of the next element required just by adding a pre-calculated offset for speed.

Readers are advised to consult a standard mathematics textbook for details of matrix arithmetic.

```
100 A$=" NOITARTSNOMED LTC  DNA YLPITLUM XIRTAM "
110 CTL(,,,0,0,1)
120 FORI=1TOLEN(A$)
130 B$=MID$(A$,I,1)
140 C=ASC(B$)AND15
150 PRINTCTL(40-I,1,COR1,,C);B$
```

```
160 NEXT
170 PRINTCTL(,3,14)" TIME IN BASIC"
180 PRINTCTL(,,15)" TIME IN MAT  "CTL(,,5)
190 XP=10
200 X1=3
210 Y1=3
220 X2=3
230 Y2=1
240 X3=1
250 Y3=3
260 DIMA(X1,Y1),B(X2,Y2),C(X3,Y3)
270 GOSUB430
280 GOSUB450
290 PRINTCTL(1,7);"--------------------------------------"
300 T1=TI
310 GOSUB470
320 PRINTCTL(15,3,14)(TI-T1)/60CTL(,7)
330 GOSUB410
340 PRINTCTL(1,12,5);"--------------------------------------"
350 T1=TI
360 MAT A=B*C
370 PRINTCTL(15,4,15)(TI-T1)/60CTL(,12)
380 GOSUB410
390 PRINTCTL(1,17,5);"--------------------------------------"
400 CTL(0,22):END
410 FORI=0TOX1:FORJ=0TOY1:PRINTCTL((J+1)*XP-10);A(I,J);:NEXT:PRINT:NEXT
420 RETURN
430 FORI=0TOX2:FORJ=0TOY2:READB(I,J):NEXT:NEXT
440 RETURN
450 FORI=0TOX3:FORJ=0TOY3:READC(I,J):NEXT:NEXT
460 RETURN
470 FORY=0TOY1
480 FORX=0TOX1
490 T=0
500 FORI=0TOY2
510 T=T+B(X,I)*C(I,Y)
520 NEXT
530 A(X,Y)=T
540 NEXT
550 NEXT
560 RETURN
570 DATA1,2
580 DATA3,4
590 DATA5,6
600 DATA7,8E-5
610 REM
620 DATA1,2,3;4
630 DATA5,6,7,8
```

Program 17. Demonstration of the MAT command and use of CTL command.

```
LOC    CODE          LINE


90AC                        .LIB MAT.COMMAND
90AC                 ;**************************
90AC                 ; 16 BIT UNSIGNED MULTIPLY
90AC                 ;**************************
90AC                 ; WAREA = N1 * N2
90AC                 ;
90AC   00 00         N1      .WOR 0
90AE   00 00         N2      .WOR 0
90B0   00 00         RESULT  .WOR 0
90B2                 ;
90B2   A9 00         MMULT   LDA #0             ;ZERO RESULT
90B4   8D B0 90              STA RESULT
```

```
LOC   CODE        LINE


90B7  8D B1 90           STA RESULT+1
90BA  AD AE 90           LDA N2         ; END IF N2=0
90BD  0D AF 90           ORA N2+1
90C0  F0 08              BEQ MMULT2
90C2  AD AC 90    MMULT1 LDA N1          ;N1 = 0 ?
90C5  0D AD 90           ORA N1+1
90C8  D0 01              BNE MMULT3
90CA  60          MMULT2 RTS
90CB  A9 01       MMULT3 LDA #1          ;IF BIT 0 OF N1
90CD  2D AC 90           AND N1         ;THEN ADD N2 TO RESULT
90D0  F0 13              BEQ MMULT4
90D2  18                 CLC            ;ADD N2 TO RESULT
90D3  AD AE 90           LDA N2
90D6  6D B0 90           ADC RESULT
90D9  8D B0 90           STA RESULT
90DC  AD AF 90           LDA N2+1
90DF  6D B1 90           ADC RESULT+1
90E2  8D B1 90           STA RESULT+1
90E5  0E AE 90    MMULT4 ASL N2          ;N2 = N2 * 2
90E8  2E AF 90           ROL N2+1
90EB  4E AD 90           LSR N1+1       ;N1 = N1 / 2
90EE  6E AC 90           ROR N1
90F1  4C C2 90           JMP MMULT1
90F4              ;
90F4              ;
90F4              ;
90F4              ;************************
90F4              ; MATRIX ARITHMETIC
90F4              ;************************
90F4              ;
90F4              ISNALF = $B113
90F4              CHRGOT = $79
90F4              CHRGET = $73
90F4  00 00       VNAME1 .WOR 0          ;VARIABLE NAMES
90F6  00          VTYPE1 .BYT 0
90F7  00 00       VNAME2 .WOR 0
90F9  00          VTYPE2 .BYT 0
90FA  00 00       VNAME3 .WOR 0
90FC  00          VTYPE3 .BYT 0
90FD              FACM   * = *+5         ;TEMP FLOATING STORE
9102              FACT   * = *+5         ;TEMP FLOATING STORE
9107  00 00       VSIZE1 .WOR 0          ;ARRAY SIZES
9109  00 00       VSIZE2 .WOR 0
910B  00 00       VSIZE3 .WOR 0
910D  00          OPTYPE .BYT 0          ;OPERAND TYPE
910E              VPTR1  = $FB
910E              VPTR2  = $FD
910E              VPTR3  = $9E
910E  00 00       VSTT1  .WOR 0
9110  00 00       VSTT2  .WOR 0
9112  00 00       VSTT3  .WOR 0
9114  00 00       T1     .WOR 0
9116  00 00       T2     .WOR 0
9118              ;
9118              ;
9118              ;
9118  8D F4 90    MAT    STA VNAME1      ;GET FIRST ARRAY
911B  20 13 B1           JSR ISNALF     ;NAME AND CHECK
911E  B0 03              BCS CHOK       ; LEGAL
9120  4C 08 AF           JMP $AF08      ; SYNTAX
9123  A9 00       CHOK   LDA #0
9125  8D F5 90           STA VNAME1+1
9128  8D F8 90           STA VNAME2+1
912B  8D FB 90           STA VNAME3+1
912E  8D F6 90           STA VTYPE1
```

```
LOC   CODE          LINE


9131  8D F9 90             STA VTYPE2
9134  8D FC 90             STA VTYPE3
9137  20 73 00             JSR CHRGET
913A  90 05                BCC CHOK1
913C  20 13 B1             JSR ISNALF
913F  90 0D                BCC EDVNA1     ;GO CHECK FOR % $ =
9141  8D F5 90      CHOK1  STA VNAME1+1
9144  20 73 00      LNE    JSR CHRGET      ;SCAN PAST REST
9147  90 FB                BCC LNE        ;OF VAR NAME
9149  20 13 B1             JSR ISNALF
914C  B0 F6                BCS LNE
914E  C9 24         EDVNA1 CMP #'$         ;CHECK FOR STRING
9150  D0 05                BNE NSTR1
9152  A2 16         TYMISE LDX #22
9154  4C 37 A4             JMP $A437      ;TYPE MISMATCH
9157  C9 25         NSTR1  CMP #'%
9159  D0 06                BNE NTINT1     ;NOT INTEGER ARRAY
915B  CE F6 90             DEC VTYPE1     ;SET TYPE FLAG TO $FF
915E  20 73 00             JSR CHRGET     ;GET NEXT CHAR
9161  C9 B2         NTINT1 CMP #$B2        ; TOKEN FOR =
9163  F0 03                BEQ FOEQ
9165  4C 08 AF             JMP $AF08      ;SYNTAX NOT =
9168  20 73 00      FOEQ   JSR CHRGET
916B  C9 28                CMP #'(        ;CHECK FOR ( EXP. )
916D  D0 16                BNE NTEXP2
916F  20 F1 AE             JSR $AEF1      ;EVAL. EXP. IN ( )
9172  A5 0D                LDA $0D        ;CHECK NUMERIC
9174  D0 DC                BNE TYMISE
9176  A2 FD                LDX #<FACM     ;FAC#1 TO FACM
9178  A0 90                LDY #>FACM
917A  20 D4 BB             JSR $BBD4
917D  A2 01                LDX #1         ; SET TYPE FLAG TO CONST
917F  8E F9 90             STX VTYPE2
9182  4C BA 91             JMP CHKOP
9185  20 13 B1      NTEXP2 JSR ISNALF      ;GET NAME
9188  B0 03                BCS CHOK2      ;CHECK LEGAL
918A  4C 08 AF             JMP $AF08      ;SYNTAX
918D  8D F7 90      CHOK2  STA VNAME2
9190  20 73 00             JSR CHRGET     ;GET SECOND CHAR
9193  90 05                BCC CHOK2A     ;NUMBER ?
9195  20 13 B1             JSR ISNALF
9198  90 0D                BCC EDVNA2     ;CHECK FOR $ %
919A  8D F8 90      CHOK2A STA VNAME2+1
919D  20 73 00      LNE2   JSR CHRGET      ;SCAN TO END
91A0  90 FB                BCC LNE2       ;OF VARIABLE NAME
91A2  20 13 B1             JSR ISNALF
91A5  B0 F6                BCS LNE2
91A7  C9 24         EDVNA2 CMP #'$         ;CHECK FOR '$'
91A9  D0 05                BNE NSTR2
91AB  A2 16                LDX #22        ; TYPE MISMATCH
91AD  4C 37 A4             JMP $A437
91B0  C9 25         NSTR2  CMP #'%         ;CHECK IF INTEGER
91B2  D0 06                BNE CHKOP
91B4  20 73 00             JSR CHRGET
91B7  CE F9 90             DEC VTYPE2     ;SET INTEGER FLAG
91BA  A2 00         CHKOP  LDX #0          ;CHECK OPERAND TYPE
91BC  8E 0D 91             STX OPTYPE
91BF  20 79 00             JSR CHRGOT     ;END STATEMENT ?
91C2  D0 03                BNE NASSIG
91C4  4C 49 92             JMP DOMAT
91C7  EE 0D 91      NASSIG INC OPTYPE
91CA  C9 AA                CMP #$AA       ;CHECK FOR ADD +
91CC  F0 11                BEQ GETV3
91CE  EE 0D 91             INC OPTYPE
91D1  C9 AB                CMP #$AB       ;CHECK FOR SUB -
```

```
LOC     CODE          LINE


91D3    F0 0A                  BEQ GETV3
91D5    EE 0D 91               INC OPTYPE
91D8    C9 AC                  CMP #$AC          ;CHECK FOR MULT *
91DA    F0 03                  BEQ GETV3
91DC    4C 08 AF               JMP $AF08         ;SYNTAX
91DF    20 73 00      GETV3    JSR CHRGET
91E2    C9 28                  CMP #'(           ;CHECK FOR ( EXP )
91E4    D0 28                  BNE NTEXP3
91E6    AD F9 90               LDA VTYPE2        ;CHECK TYPE2 FOR
91E9    C9 01                  CMP #1            ;BEING  CONSTANT
91EB    D0 03                  BNE BEXPOK
91ED    4C 08 AF               JMP $AF08         ;SYNTAX
91F0    20 F1 AE      BEXPOK   JSR $AEF1          ;EVAL EXP
91F3    A5 0D                  LDA $0D
91F5    F0 03                  BEQ NUMOK
91F7    4C 52 91               JMP TYMISE        ;TYPE MISMATCH
91FA    A2 FD         NUMOK    LDX #<FACM         ;FAC#1 TO FACM
91FC    A0 90                  LDY #>FACM
91FE    20 D4 BB               JSR $BBD4
9201    A9 01                  LDA #1            ;SET TYPE FLAG TO CONST
9203    8D FC 90               STA VTYPE3
9206    20 79 00               JSR CHRGOT        ;END OF STATEMENT ?
9209    F0 3E                  BEQ DOMAT
920B    4C 08 AF      SYNTE    JMP $AF08          ;SYNTAX
920E    20 13 B1      NTEXP3   JSR ISNALF         ;GET ARRAY NAME
9211    90 F8                  BCC SYNTE         ;SYNTAX ERROR
9213    8D FA 90               STA VNAME3
9216    20 73 00               JSR CHRGET
9219    F0 2E                  BEQ DOMAT         ; : OR END OF LINE
921B    90 05                  BCC CHOK3
921D    20 13 B1               JSR ISNALF
9220    90 0F                  BCC EDVNA3
9222    8D FB 90      CHOK3    STA VNAME3+1
9225    20 73 00      LNE3     JSR CHRGET
9228    F0 1F                  BEQ DOMAT
922A    90 F9                  BCC LNE3
922C    20 13 B1               JSR ISNALF
922F    B0 F4                  BCS LNE3
9231    C9 24         EDVNA3   CMP #'$            ;IS IT A STRING
9233    D0 05                  BNE NSTR3
9235    A2 16                  LDX #22
9237    4C 37 A4               JMP $A437
923A    C9 25         NSTR3    CMP #'%            ;IS IT INTEGER
923C    D0 08                  BNE NTINT3
923E    CE FC 90               DEC VTYPE3
9241    20 73 00               JSR CHRGET        ;NEXT CHAR
9244    F0 03                  BEQ DOMAT
9246    4C 08 AF      NTINT3   JMP $AF08          ;SYNTAX
9249    AD F6 90      DOMAT    LDA VTYPE1         ;FIND ARRAY 1
924C    F0 10                  BEQ V1REAL
924E    A9 80                  LDA #128          ;SET HI BITS ARRAY NAME
9250    0D F4 90               ORA VNAME1
9253    8D F4 90               STA VNAME1
9256    A9 80                  LDA #128
9258    0D F5 90               ORA VNAME1+1
925B    8D F5 90               STA VNAME1+1
925E    20 78 94      V1REAL   JSR FINDAR         ;FIND ARRAY ADDR
9261    8E 07 91               STX VSIZE1
9264    8C 08 91               STY VSIZE1+1
9267    A5 FB                  LDA VPTR1         ;STORE IT
9269    8D 0E 91               STA VSTT1
926C    A5 FC                  LDA VPTR1+1
926E    8D 0F 91               STA VSTT1+1
9271    AD F9 90               LDA VTYPE2
9274    C9 01                  CMP #1
```

```
LOC   CODE       LINE

9276  F0 2A              BEQ GAR3        ;EXPRESSION
9278  AD F9 90           LDA VTYPE2      ;SET UP ARRAY NAME 2
927B  29 80              AND #$80        ;FOR SEARCH ROUTINE
927D  8D 14 91           STA T1
9280  0D F7 90           ORA VNAME2
9283  8D F4 90           STA VNAME1
9286  AD F8 90           LDA VNAME2+1
9289  0D 14 91           ORA T1
928C  8D F5 90           STA VNAME1+1
928F  20 78 94           JSR FINDAR      ;FIND ADDRESS ARRAY 2
9292  8E 09 91           STX VSIZE2
9295  8C 0A 91           STY VSIZE2+1
9298  A5 FB              LDA VPTR1
929A  8D 10 91           STA VSTT2
929D  A5 FC              LDA VPTR1+1
929F  8D 11 91           STA VSTT2+1
92A2  AD 0D 91   GAR3    LDA OPTYPE       ;ARRAY 3  ?
92A5  F0 31              BEQ DOMATA      ;NO ARRAY 3
92A7  AD FC 90           LDA VTYPE3
92AA  C9 01              CMP #1          ;IS IT A CONSTANT
92AC  F0 2A              BEQ DOMATA      ;YES
92AE  29 80              AND #$80        ; IS ARRAY 3 INTEGER
92B0  8D 14 91           STA T1
92B3  AD FA 90           LDA VNAME3
92B6  0D 14 91           ORA T1
92B9  8D F4 90           STA VNAME1
92BC  AD FB 90           LDA VNAME3+1
92BF  0D 14 91           ORA T1
92C2  8D F5 90           STA VNAME1+1
92C5  20 78 94           JSR FINDAR      ;FIND ARRAY 3
92C8  8E 0B 91           STX VSIZE3
92CB  8C 0C 91           STY VSIZE3+1
92CE  A5 FB              LDA VPTR1
92D0  8D 12 91           STA VSTT3
92D3  A5 FC              LDA VPTR1+1
92D5  8D 13 91           STA VSTT3+1
92D8  AD 0D 91   DOMATA LDA OPTYPE        ;SET A JUMP VECTOR
92DB  0A                 ASL A           ;FOR OPERATION
92DC  AA                 TAX
92DD  BD EE 92           LDA OPJTAB,X
92E0  8D EC 92           STA OPJMP
92E3  BD EF 92           LDA OPJTAB+1,X
92E6  8D ED 92           STA OPJMP+1
92E9  6C EC 92           JMP (OPJMP)
92EC             ;
92EC  00 00      OPJMP   .WOR 0           ;JUMP VECTOR
92EE  F6 92      OPJTAB .WOR ASSGN        ;JUMP TABLE
92F0  01 95              .WOR ADDSUB
92F2  01 95              .WOR ADDSUB
92F4  6A 96              .WOR MULT
92F6             ; ***   MAT AA = C
92F6  A9 01      ASSGN   LDA #1
92F8  CD F9 90           CMP VTYPE2
92FB  F0 03              BEQ ASSIC
92FD  4C 63 93           JMP ASARAR
9300  A2 05      ASSIC   LDX #5           ;ARRAY =CONSTANT
9302  AD F6 90           LDA VTYPE1
9305  F0 16              BEQ ASSR1
9307  A9 FD              LDA #<FACM      ;FACM TO FAC#1
9309  A0 90              LDY #>FACM
930B  20 A2 BB           JSR $BBA2
930E  20 BF B1           JSR $B1BF       ;FLOAT TO FIXED
9311  A5 64              LDA $64         ;STORE INT IN FACM
9313  8D FD 90           STA FACM
9316  A5 65              LDA $65
```

```
LOC   CODE        LINE


9318  8D FE 90            STA FACM+1
931B  A2 02               LDX #2
931D  8E F9 90    ASSR1   STX VTYPE2       ;STORE ELEMENT LENGTH
9320  A9 00               LDA #0          ;CALC NUMBER OF ELEMENTS
9322  8D AD 90            STA N1+1
9325  8D AF 90            STA N2+1
9328  AD 07 91            LDA VSIZE1
932B  8D AC 90            STA N1
932E  AD 08 91            LDA VSIZE1+1
9331  8D AE 90            STA N2
9334  20 B2 90            JSR MMULT       ;RESULT =N1 * N2
9337  20 C0 95            JSR TRPT1       ;COPY POINTER TO ZERO PAGE
933A  A0 00               LDY #0
933C  A2 00       ASLOOP  LDX #0           ;FACM TO ARRAY
933E  BD FD 90    ASLOP   LDA FACM,X
9341  91 FB               STA (VPTR1),Y
9343  E8                  INX
9344  E6 FB               INC VPTR1
9346  D0 02               BNE ASNC
9348  E6 FC               INC VPTR1+1
934A  EC F9 90    ASNC    CPX VTYPE2
934D  D0 EF               BNE ASLOP
934F  AD B0 90            LDA RESULT
9352  D0 03               BNE ASNC9
9354  CE B1 90            DEC RESULT+1
9357  CE B0 90    ASNC9   DEC RESULT       ;ARRAY FILLED ?
935A  AD B0 90            LDA RESULT
935D  0D B1 90            ORA RESULT+1
9360  D0 DA               BNE ASLOOP
9362  60                  RTS
9363                  ;
9363  A2 05       ASARAR  LDX #5           ;SET VAR LENGTH
9365  AD F6 90            LDA VTYPE1
9368  F0 02               BEQ ASR1R
936A  A2 02               LDX #2
936C  8E F6 90    ASR1R   STX VTYPE1
936F  A2 05               LDX #5
9371  AD F9 90            LDA VTYPE2
9374  F0 02               BEQ ASR2R
9376  A2 02               LDX #2
9378  8E F9 90    ASR2R   STX VTYPE2
937B  AD 07 91            LDA VSIZE1      ;COMPARE ARRAY SIZES
937E  CD 09 91            CMP VSIZE2
9381  F0 05               BEQ ASRSOK
9383  A2 12       ASRSUB  LDX #$12         ;BAD SUBSCRIPT ERROR
9385  4C 37 A4            JMP $A437
9388  AD 08 91    ASRSOK  LDA VSIZE1+1
938B  CD 0A 91            CMP VSIZE2+1
938E  D0 F3               BNE ASRSUB      ; ERROR
9390  AD F6 90            LDA VTYPE1      ;ARRAYS SAME TYPE ?
9393  CD F9 90            CMP VTYPE2
9396  D0 5A               BNE ASRIR       ;NO
9398  A9 00               LDA #0          ;CALC SIZE OF ARRAYS
939A  8D AD 90            STA N1+1
939D  8D AF 90            STA N2+1
93A0  AD 07 91            LDA VSIZE1
93A3  8D AC 90            STA N1
93A6  AD 08 91            LDA VSIZE1+1
93A9  8D AE 90            STA N2
93AC  20 B2 90            JSR MMULT
93AF  AD B0 90            LDA RESULT
93B2  8D AC 90            STA N1
93B5  AD B1 90            LDA RESULT+1
93B8  8D AD 90            STA N1+1
93BB  AD F6 90            LDA VTYPE1
```

```
LOC   CODE        LINE

93BE  8D AE 90           STA N2
93C1  A9 00              LDA #0
93C3  8D AF 90           STA N2+1
93C6  20 B2 90           JSR MMULT
93C9  20 B6 95           JSR TRPT2        ;SET POINTERS TO ARRAYS
93CC  A0 00              LDY #0
93CE  B1 FD       ASSTLO LDA (VPTR2),Y     ;BLOCK MOVE OF
93D0  91 FB              STA (VPTR1),Y    ;LENGTH IN RESULT
93D2  E6 FB              INC VPTR1
93D4  D0 02              BNE ASSTN1
93D6  E6 FC              INC VPTR1+1
93D8  E6 FD       ASSTN1 INC VPTR2
93DA  D0 02              BNE ASSTN2
93DC  E6 FE              INC VPTR2+1
93DE  AD B0 90    ASSTN2 LDA RESULT
93E1  D0 03              BNE ASSTN3
93E3  CE B1 90           DEC RESULT+1
93E6  CE B0 90    ASSTN3 DEC RESULT
93E9  AD B0 90           LDA RESULT
93EC  0D B1 90           ORA RESULT+1
93EF  D0 DD              BNE ASSTLO
93F1  60                 RTS
93F2  A9 00       ASRIR  LDA #0
93F4  8D AD 90           STA N1+1
93F7  8D AF 90           STA N2+1
93FA  AD 07 91           LDA VSIZE1
93FD  8D AC 90           STA N1           ;CALC NUMBER OF ELEMENTS
9400  AD 08 91           LDA VSIZE1+1
9403  8D AE 90           STA N2
9406  20 B2 90           JSR MMULT
9409  20 B6 95           JSR TRPT2
940C  A0 00       ASRLOP LDY #0
940E  A2 00              LDX #0           ;ARRAY ELEMENT TO FACM
9410  B1 FD       ASRLP1 LDA (VPTR2),Y
9412  9D FD 90           STA FACM,X
9415  E6 FD              INC VPTR2
9417  D0 02              BNE ASRNC2
9419  E6 FE              INC VPTR2+1
941B  E8          ASRNC2 INX
941C  EC F9 90           CPX VTYPE2
941F  D0 EF              BNE ASRLP1
9421  E0 05              CPX #5
9423  D0 17              BNE ASRITR
9425  A9 FD              LDA #<FACM       ;FACM TO FAC#1
9427  A0 90              LDY #>FACM
9429  20 A2 BB           JSR $BBA2
942C  20 BF B1           JSR $B1BF        ;FLOAT TO FIXED
942F  A5 64              LDA $64
9431  8D FD 90           STA FACM
9434  A5 65              LDA $65
9436  8D FE 90           STA FACM+1
9439  4C 4C 94           JMP ASRTM        ;FACM TO ARRAY
943C  AD FD 90    ASRITR LDA FACM
943F  AC FE 90           LDY FACM+1
9442  20 91 B3           JSR $B391        ;FIXED TO FLOAT
9445  A2 FD              LDX #<FACM       ;FAC#1 TO FACM
9447  A0 90              LDY #>FACM
9449  20 D4 BB           JSR $BBD4
944C  A0 00       ASRTM  LDY #0
944E  A2 00              LDX #0
9450  BD FD 90    ASRTM1 LDA FACM,X
9453  91 FB              STA (VPTR1),Y
9455  E8                 INX
9456  E6 FB              INC VPTR1
9458  D0 02              BNE ASRNC1
```

```
LOC   CODE        LINE


945A  E6 FC                INC VPTR1+1
945C  EC F6 90    ASRNC1 CPX VTYPE1
945F  D0 EF                BNE ASRTM1
9461  AD B0 90             LDA RESULT
9464  D0 03                BNE ASRTM3
9466  CE B1 90             DEC RESULT+1
9469  CE B0 90    ASRTM3 DEC RESULT
946C  AD B0 90             LDA RESULT
946F  0D B1 90             ORA RESULT+1
9472  F0 03                BEQ ASREXT
9474  4C 0C 94             JMP ASRLOP
9477  60          ASREXT RTS
9478              ;
9478              ; FIND ARRAY
9478  A5 2F       FINDAR LDA $2F          ;START OF ARRAYS
947A  85 FB                STA VPTR1
947C  A5 30                LDA $30
947E  85 FC                STA VPTR1+1
9480  A5 FB       FALOOP LDA VPTR1        ;CMP. END OF ARRAYS
9482  C5 31                CMP $31
9484  D0 0B                BNE FACONT
9486  A5 FC                LDA VPTR1+1
9488  C5 32                CMP $32
948A  D0 05                BNE FACONT
948C  A2 12                LDX #$12        ;BAD SUBSCRIPT ERROR
948E  20 37 A4             JSR $A437
9491  A0 00       FACONT LDY #0
9493  B1 FB                LDA (VPTR1),Y   ;FIRST CHAR OF NAME
9495  C8                   INY
9496  CD F4 90             CMP VNAME1
9499  D0 07                BNE FANAR       ;TRY NEXT ARRAY
949B  B1 FB                LDA (VPTR1),Y
949D  CD F5 90             CMP VNAME1+1
94A0  F0 1D                BEQ FAGETS      ; GET ARRAY DATA
94A2  C8          FANAR  INY              ;FIND NEXT ARRAY
94A3  B1 FB                LDA (VPTR1),Y
94A5  8D 14 91             STA T1
94A8  C8                   INY
94A9  B1 FB                LDA (VPTR1),Y
94AB  18                   CLC
94AC  65 FC                ADC VPTR1+1
94AE  85 FC                STA VPTR1+1
94B0  AD 14 91             LDA T1
94B3  18                   CLC
94B4  65 FB                ADC VPTR1
94B6  85 FB                STA VPTR1
94B8  90 02                BCC FANC
94BA  E6 FC                INC VPTR1+1
94BC  4C 80 94    FANC   JMP FALOOP
94BF  A9 01       FAGETS LDA #1           ;GET ARRAY DATA
94C1  8D 15 91             STA T1+1
94C4  C8                   INY
94C5  C8                   INY
94C6  C8                   INY
94C7  B1 FB                LDA (VPTR1),Y
94C9  C9 03                CMP #3
94CB  30 05                BMI FANDOK
94CD  A2 12       FAE1   LDX #$12         ;ERROR MORE THAN 2 DIM
94CF  4C 37 A4             JMP $A437
94D2  AA          FANDOK TAX
94D3  C8                   INY
94D4  B1 FB                LDA (VPTR1),Y
94D6  D0 F5                BNE FAE1        ;FIRST DIM TOO BIG
94D8  C8                   INY
94D9  B1 FB                LDA (VPTR1),Y
```

```
LOC   CODE        LINE


94DB  8D 14 91            STA T1
94DE  8A                  TXA
94DF  CA                  DEX
94E0  F0 0B               BEQ FAEX       ;ONE DIM ARRAY
94E2  C8                  INY
94E3  B1 FB               LDA (VPTR1),Y
94E5  D0 E6               BNE FAE1       ;SECOND DIM TOO BIG
94E7  C8                  INY
94E8  B1 FB               LDA (VPTR1),Y
94EA  8D 15 91            STA T1+1
94ED  C8          FAEX    INY
94EE  98                  TYA
94EF  18                  CLC
94F0  65 FB               ADC VPTR1
94F2  85 FB               STA VPTR1
94F4  A5 FC               LDA VPTR1+1
94F6  69 00               ADC #0
94F8  85 FC               STA VPTR1+1
94FA  AE 14 91            LDX T1
94FD  AC 15 91            LDY T1+1
9500  60                  RTS
9501              ;
9501  20 81 95    ADDSUB  JSR ORDER       ;PUT CONST LAST
9504  AD 07 91            LDA VSIZE1     ;CHECK ARRAY SIZES
9507  8D AC 90            STA N1
950A  CD 09 91            CMP VSIZE2
950D  D0 22               BNE ADBADS
950F  AD 08 91            LDA VSIZE1+1
9512  8D AE 90            STA N2
9515  CD 0A 91            CMP VSIZE2+1
9518  D0 17               BNE ADBADS
951A  AD F9 90            LDA VTYPE2     ;V2 CONSTANT ?
951D  C9 01               CMP #1
951F  F0 15               BEQ ABSC
9521  AD 09 91            LDA VSIZE2     ;V3 IS ARRAY
9524  CD 0B 91            CMP VSIZE3
9527  D0 08               BNE ADBADS
9529  AD 0A 91            LDA VSIZE2+1
952C  CD 0C 91            CMP VSIZE3+1
952F  F0 05               BEQ ABSC
9531  A2 12       ADBADS  LDX #$12        ;BAD SUBSCRIPT
9533  4C 37 A4            JMP $A437
9536  20 AC 95    ABSC    JSR TRPT3       ;COPY POINTER TO Z PAGE
9539  A9 00               LDA #0         ;CALC NO. OF ELEMENTS
953B  8D AD 90            STA N1+1
953E  8D AF 90            STA N2+1
9541  20 B2 90            JSR MMULT
9544  20 CB 95    ABSLOP  JSR V2TOT2      ;V2 TO (T2)
9547  20 13 96            JSR V3TOF1     ;V2 TO FAC#1
954A  AD 16 91            LDA T2
954D  AC 17 91            LDY T2+1
9550  AE 0D 91            LDX OPTYPE
9553  E0 01               CPX #1
9555  D0 06               BNE DOSUB
9557  20 67 B8            JSR $B867      ; (A.Y) + FAC#1
955A  4C 6A 95            JMP ABFA
955D  E0 02       DOSUB   CPX #2
955F  D0 06               BNE DOMULT
9561  20 50 B8            JSR $B850      ;(A.Y)-FAC#1
9564  4C 6A 95            JMP ABFA
9567  20 28 BA    DOMULT  JSR $BA28       ;(A.Y) * FAC#1
956A  20 3E 96    ABFA    JSR F1TOV1      ;FAC#1 TO V1
956D  AD B0 90            LDA RESULT     ;CHECK ALL DONE
9570  D0 03               BNE ABNC
9572  CE B1 90            DEC RESULT+1
```

```
LOC   CODE        LINE


9575  CE B0 90    ABNC   DEC RESULT
9578  AD B0 90           LDA RESULT
957B  0D B1 90           ORA RESULT+1
957E  D0 C4              BNE ABSLOP
9580  60                 RTS
9581                ;
9581  AD FC 90    ORDER  LDA VTYPE3      ;V2 CONST
9584  C9 01              CMP #1
9586  D0 23              BNE ADV2NC
9588  AD F9 90           LDA VTYPE2     ;SWOT V2 & V3
958B  8D FC 90           STA VTYPE3
958E  AD 09 91           LDA VSIZE2
9591  8D 0B 91           STA VSIZE3
9594  AD 0A 91           LDA VSIZE2+1
9597  8D 0C 91           STA VSIZE3+1
959A  AD 10 91           LDA VSTT2
959D  8D 12 91           STA VSTT3
95A0  AD 11 91           LDA VSTT2+1
95A3  8D 13 91           STA VSTT3+1
95A6  A9 01              LDA #1
95A8  8D F9 90           STA VTYPE2
95AB  60          ADV2NC RTS
95AC                ;
95AC  AD 12 91    TRPT3  LDA VSTT3       ; COPY POINTERS TO
95AF  85 9E              STA VPTR3      ;ZERO PAGE
95B1  AD 13 91           LDA VSTT3+1
95B4  85 9F              STA VPTR3+1
95B6  AD 10 91    TRPT2  LDA VSTT2
95B9  85 FD              STA VPTR2
95BB  AD 11 91           LDA VSTT2+1
95BE  85 FE              STA VPTR2+1
95C0  AD 0E 91    TRPT1  LDA VSTT1
95C3  85 FB              STA VPTR1
95C5  AD 0F 91           LDA VSTT1+1
95C8  85 FC              STA VPTR1+1
95CA  60                 RTS
95CB                ;
95CB  AD F9 90    V2TOT2 LDA VTYPE2      ;V2 TO FAC#2
95CE  F0 0D              BEQ V2RA
95D0  30 23              BMI V2INT
95D2  A9 FD              LDA #<FACM     ;FACM TO FAC#2
95D4  A0 90              LDY #>FACM
95D6  8D 16 91           STA T2
95D9  8C 17 91           STY T2+1
95DC  60                 RTS
95DD  A5 FD       V2RA   LDA VPTR2       ;V2 TO FAC#2
95DF  A4 FE              LDY VPTR2+1
95E1  8D 16 91           STA T2
95E4  8C 17 91           STY T2+1
95E7  A9 05              LDA #5
95E9  18          V2BPT  CLC             ;BUMP VPTR2
95EA  65 FD              ADC VPTR2
95EC  85 FD              STA VPTR2
95EE  A5 FE              LDA VPTR2+1
95F0  69 00              ADC #0
95F2  85 FE              STA VPTR2+1
95F4  60                 RTS
95F5  A0 00       V2INT  LDY #0          ;FIXED TO FLOAT
95F7  B1 FD              LDA (VPTR2),Y  ;THEN FAC#1 TO FAC#2
95F9  AA                 TAX
95FA  C8                 INY
95FB  B1 FD              LDA (VPTR2),Y
95FD  A8                 TAY
95FE  8A                 TXA
95FF  20 91 B3           JSR $B391      ;FIXED TO FLOAT
```

```
LOC     CODE          LINE


9602    A2 02                LDX #<FACT      ;FAC#1 TO FACT
9604    8E 16 91             STX T2
9607    A0 91                LDY #>FACT
9609    8C 17 91             STY T2+1
960C    20 D4 BB             JSR $BBD4
960F    A9 02                LDA #2
9611    D0 D6                BNE V2BPT       ;GO BUMP VPTR2
9613    AD FC 90      V3TOF1 LDA VTYPE3
9616    D0 15                BNE V3INT
9618    A5 9E                LDA VPTR3       ;V3 TO FAC#1
961A    A4 9F                LDY VPTR3+1
961C    20 A2 BB             JSR $BBA2
961F    A9 05                LDA #5
9621    18            V3BPT  CLC              ;BUMP VPTR3
9622    65 9E                ADC VPTR3
9624    85 9E                STA VPTR3
9626    A5 9F                LDA VPTR3+1
9628    69 00                ADC #0
962A    85 9F                STA VPTR3+1
962C    60                   RTS
962D    A0 00         V3INT  LDY #0           ;GET V3
962F    B1 9E                LDA (VPTR3),Y
9631    AA                   TAX
9632    C8                   INY
9633    B1 9E                LDA (VPTR3),Y
9635    A8                   TAY
9636    8A                   TXA
9637    20 91 B3             JSR $B391       ;FIXED TO FLOAT
963A    A9 02                LDA #2
963C    D0 E3                BNE V3BPT       ;GO BUMP VPTR3
963E    AD F6 90      F1TOV1 LDA VTYPE1       ;FAC#1 TO V1
9641    D0 15                BNE V1INT
9643    A6 FB                LDX VPTR1
9645    A4 FC                LDY VPTR1+1
9647    20 D4 BB             JSR $BBD4
964A    A9 05                LDA #5
964C    18            V1BPT  CLC              ;BUMP VPTR1
964D    65 FB                ADC VPTR1
964F    85 FB                STA VPTR1
9651    A5 FC                LDA VPTR1+1
9653    69 00                ADC #0
9655    85 FC                STA VPTR1+1
9657    60                   RTS
9658    20 BF B1      V1INT  JSR $B1BF        ;FLOAT TO INT
965B    A0 00                LDY #0
965D    A5 64                LDA $64
965F    91 FB                STA (VPTR1),Y
9661    A5 65                LDA $65
9663    C8                   INY
9664    91 FB                STA (VPTR1),Y
9666    A9 02                LDA #2
9668    D0 E2                BNE V1BPT
966A                  ;
966A    AD F9 90      MULT   LDA VTYPE2       ;CHECK FOR MULT.
966D    C9 01                CMP #1          ;ARRAY BY CONSTANT
966F    D0 03                BNE MERR
9671    4C 01 95      GADS   JMP ADDSUB
9674    AD FC 90      MERR   LDA VTYPE3
9677    C9 01                CMP #1
9679    F0 F6                BEQ GADS
967B    AD 08 91             LDA VSIZE1+1    ;CHECK ARRAY DIM.
967E    CD 0A 91             CMP VSIZE2+1
9681    D0 30                BNE AAERR
9683    AD 07 91             LDA VSIZE1      ;CHECK NOT SAME ARRAYS
9686    CD 0B 91             CMP VSIZE3
```

```
LOC   CODE        LINE


9689  D0 28              BNE AAERR
968B  AD 09 91           LDA VSIZE2
968E  CD 0C 91           CMP VSIZE3+1
9691  D0 20              BNE AAERR
9693  AD 0E 91           LDA VSTT1
9696  CD 10 91           CMP VSTT2
9699  D0 08              BNE NSARR0
969B  AD 0F 91           LDA VSTT1+1
969E  CD 11 91           CMP VSTT2+1
96A1  F0 10              BEQ AAERR
96A3  AD 0E 91    NSARR0 LDA VSTT1
96A6  CD 12 91           CMP VSTT3
96A9  D0 0D              BNE AASOK
96AB  AD 0F 91           LDA VSTT1+1
96AE  CD 13 91           CMP VSTT3+1
96B1  D0 05              BNE AASOK
96B3  A2 12       AAERR  LDX #$12         ;BAD SUBSCRIPT ERROR
96B5  4C 37 A4           JMP $A437
96B8  20 AC 95    AASOK  JSR TRPT3        ;COPY POINTERS TO Z. P.
96BB  A9 00              LDA #0
96BD  8D AD 90           STA N1+1
96C0  8D AF 90           STA N2+1
96C3  A9 01              LDA #1
96C5  8D AA 97           STA ROW
96C8  8D A9 97           STA NROW
96CB  8D AB 97           STA COL
96CE  A9 05              LDA #5          ;CALC LENGTH OF V2 ROW
96D0  AE F9 90           LDX VTYPE2      ; - 1 ELEMENT
96D3  F0 02              BEQ AA2R
96D5  A9 02              LDA #2
96D7  8D AC 90    AA2R   STA N1
96DA  8D 14 91           STA T1
96DD  AE 0A 91           LDX VSIZE2+1
96E0  CA                 DEX
96E1  8A                 TXA
96E2  8D AE 90           STA N2
96E5  20 B2 90           JSR MMULT
96E8  AD B0 90           LDA RESULT      ;STORE IT IN LLV2
96EB  8D AC 97           STA LLV2
96EE  AD B1 90           LDA RESULT+1
96F1  8D AD 97           STA LLV2+1
96F4  18          AALOOP CLC              ;MAIN LOOP
96F5  AD 10 91           LDA VSTT2       ;SET V2 COL. PTR. TO NEXT
96F8  85 FD              STA VPTR2
96FA  6D 14 91           ADC T1          ;COL OF V2
96FD  8D AE 97           STA V2COLP
9700  AD 11 91           LDA VSTT2+1
9703  85 FE              STA VPTR2+1
9705  69 00              ADC #0
9707  8D AF 97           STA V2COLP+1
970A  A9 00       AALOP  LDA #0           ;ZERO ROW COL TOTAL
970C  8D FD 90           STA FACM
970F  8D FE 90           STA FACM+1
9712  8D FF 90           STA FACM+2
9715  8D 00 91           STA FACM+3
9718  8D 01 91           STA FACM+4
971B  20 CB 95    AAMRC  JSR V2TOT2       ;GET V2
971E  20 13 96           JSR V3TOF1      ;GET V1
9721  AD 16 91           LDA T2
9724  AC 17 91           LDY T2+1
9727  20 28 BA           JSR $BA28       ;(A.Y) * FAC#1
972A  A9 FD              LDA #<FACM
972C  A0 90              LDY #>FACM
972E  20 67 B8           JSR $B867       ;(A.Y) + FAC#1
9731  AD AA 97           LDA ROW
```

```
LOC   CODE        LINE

9734  CD 09 91           CMP VSIZE2
9737  F0 1C              BEQ ENDCOL
9739  EE AA 97           INC ROW
973C  A2 FD              LDX #<FACM
973E  A0 90              LDY #>FACM
9740  20 D4 BB           JSR $BBD4       ;FAC#1 TO (X.Y)
9743  A5 FD              LDA VPTR2       ;V2 PTR DOWN 1 ROW
9745  18                 CLC
9746  6D AC 97           ADC LLV2
9749  85 FD              STA VPTR2
974B  A5 FE              LDA VPTR2+1
974D  6D AD 97           ADC LLV2+1
9750  85 FE              STA VPTR2+1
9752  4C 1B 97           JMP AAMRC       ;GET NEXT 2 ELEMENTS
9755  20 3E 96    ENDCOL JSR F1TOV1       ;FAC#1 (SUM) TO V1
9758  A9 01              LDA #1          ;FIRST ROW
975A  8D AA 97           STA ROW
975D  AD AB 97           LDA COL
9760  CD 0A 91           CMP VSIZE2+1
9763  F0 26              BEQ ENDROW
9765  AD 12 91           LDA VSTT3       ;SET V2 PTR. TO START CURRENT
9768  85 9E              STA VPTR3       ;ROW
976A  AD 13 91           LDA VSTT3+1
976D  85 9F              STA VPTR3+1
976F  EE AB 97           INC COL
9772  18                 CLC
9773  AD AE 97           LDA V2COLP
9776  85 FD              STA VPTR2
9778  6D 14 91           ADC T1
977B  8D AE 97           STA V2COLP
977E  AD AF 97           LDA V2COLP+1
9781  85 FE              STA VPTR2+1
9783  69 00              ADC #0
9785  8D AF 97           STA V2COLP+1
9788  4C 0A 97           JMP AALOP
978B  AD A9 97    ENDROW LDA NROW         ;ALL ROWS DONE ?
978E  CD 07 91           CMP VSIZE1
9791  D0 01              BNE NEAA
9793  60                 RTS             ; ALL DONE
9794  A5 9E       NEAA   LDA VPTR3
9796  8D 12 91           STA VSTT3
9799  A5 9F              LDA VPTR3+1
979B  8D 13 91           STA VSTT3+1
979E  EE A9 97           INC NROW
97A1  A9 01              LDA #1          ;FIRST COL.
97A3  8D AB 97           STA COL
97A6  4C F4 96           JMP AALOOP      ;GO NEXT ROW FIRST COL.
97A9  00          NROW   .BYT 0
97AA  00          ROW    .BYT 0
97AB  00          COL    .BYT 0
97AC  00 00       LLV2   .WOR 0
97AE  00 00       V2COLP .WOR 0
97B0                     .END
```

MERGE

Abbreviated entry: M(shift)E

Affected Basic abbreviations: None

Token: Hex $EE,$12 Decimal 238,18

Modes: Direct and program

Recommended mode: Direct only

Purpose: To merge a Basic program from disk into the current Basic program in memory.

Syntax: MERGE filename, d – where d is the device number (disk only).

Errors: Illegal device – if the device number specified is less than eight
Missing filename – if a null filename is specified
File not found – if file does not exist
Device not present – if no disk drive is connected
File open error – if ten files are already open
Disk errors – at the end, the disk error channel is read and displayed

Use: Merge is used to combine two Basic programs in memory. Each line of the program on disk is read in until the zero byte is reached, and then stored in the input buffer. The Basic routine to enter a line is then called and the line is entered at the correct place. *Note:* If a line number of the program to MERGE is the same as an existing line number, the MERGEd line will replace it.

Routine entry point: $97B0

Routine operation: The filename and device are read in and checked for missing filename and illegal device. If both checks are OK, the file is opened and the message MERGING is displayed. Each line is then read into the input buffer and entered using the Basic routine to do so. When the file is completed it is closed, and the disk error channel is read and displayed.

```
LOC   CODE        LINE


97B0                        .LIB MERGE
97B0  20 6F 98    MERGE     JSR DPARS        ; GET FILE PARAMETERS
97B3  A9 62                 LDA #<MRGMES    ; DISPLAY MERGE MESSAGE
97B5  A0 98                 LDY #>MRGMES
97B7  20 1E AB              JSR $AB1E
97BA  20 C1 F5              JSR $F5C1       ; DISPLAY FILENAME
97BD  AD 02 03              LDA $0302       ; SAVE BASIC WARM START
97C0  8D 6D 98              STA MERGST      ; LINK
97C3  AD 03 03              LDA $0303
97C6  8D 6E 98              STA MERGST+1
97C9  A9 0E                 LDA #$0E        ; FIND FILE NUMBER
97CB  20 A3 8A              JSR GETN1
97CE  85 B8                 STA $B8
97D0  8D 61 98              STA FILENO
97D3  A9 00                 LDA #$00
97D5  85 B9                 STA $B9
97D7  20 C0 FF              JSR $FFC0       ; OPEN FILE
97DA  AE 61 98              LDX FILENO
97DD  20 C6 FF              JSR $FFC6       ; SET FILE TO INPUT
97E0  A9 60                 LDA #<MERGRT
97E2  8D 2C 03              STA $032C
97E5  A9 98                 LDA #>MERGRT    ; SET 'RESET INPUT'
```

```
LOC   CODE          LINE


97E7  8D 2D 03             STA $032D      ; TO A RTS
97EA  A9 38                LDA #<MERG04
97EC  8D 02 03             STA $0302
97EF  A9 98                LDA #>MERG04   ; SET BASIC WARM START
97F1  8D 03 03             STA $0303      ; TO MERG04
97F4  20 CF FF             JSR $FFCF      ; INPUT 2 BYTE LOAD
97F7  20 CF FF             JSR $FFCF      ; ADDRESS
97FA  20 CF FF      MERG02 JSR $FFCF       ; INPUT NEXT LINE
97FD  85 14                STA $14        ; POINTERS AND
97FF  20 CF FF             JSR $FFCF      ; CHECK FOR ZERO
9802  85 15                STA $15        ; (END OF BASIC PROGRAM)
9804  05 14                ORA $14
9806  F0 33                BEQ MERG05
9808  A5 90                LDA $90        ; CHECK STATUS
980A  D0 2F                BNE MERG05
980C  20 CF FF             JSR $FFCF      ; INPUT LINE NUMBER
980F  85 14                STA $14        ; AND STORE IN $14 & $15
9811  20 CF FF             JSR $FFCF
9814  85 15                STA $15
9816  A0 00                LDY #$00
9818  20 CF FF      MERG03 JSR $FFCF       ; INPUT LINE AND
981B  99 00 02             STA $0200,Y    ; STORE IN INPUT
981E  A6 C5                LDX $C5        ; BUFFER
9820  E0 3F                CPX #63
9822  F0 17                BEQ MERG05
9824  C8                   INY
9825  C9 00                CMP #$00
9827  D0 EF                BNE MERG03     ; END OF LINE? NO.
9829  98                   TYA            ; YES
982A  18                   CLC
982B  69 04                ADC #$04
982D  85 0B                STA $0B
982F  A5 90                LDA $90        ; CHECK STATUS
9831  D0 08                BNE MERG05
9833  A4 0B                LDY $0B
9835  4C A4 A4             JMP $A4A4      ; MERGE LINE
9838  4C FA 97      MERG04 JMP MERG02      ; DO NEXT LINE
983B  AD 6D 98      MERG05 LDA MERGST      ; RESET BASIC WARM
983E  8D 02 03             STA $0302      ; START
9841  AD 6E 98             LDA MERGST+1
9844  8D 03 03             STA $0303
9847  A9 2F                LDA #$2F       ; AND 'RESET DEFAULT I/O'
9849  8D 2C 03             STA $032C
984C  A9 F3                LDA #$F3
984E  8D 2D 03             STA $032D
9851  AD 61 98             LDA FILENO
9854  20 C3 FF             JSR $FFC3      ; CLOSE FILE
9857  20 CC FF             JSR $FFCC      ; RESET DEFAULT I/O
985A  20 55 8A             JSR DISK01     ; DISPLAY ERROR CHANNEL
985D  4C 74 A4             JMP $A474      ; JUMP TO READY
9860  60            MERGRT RTS
9861  00            FILENO .BYT 0
9862  91            MRGMES .BYT $91,'MERGING: ',$00
9863  4D 45
986C  00
986D  00 00         MERGST .WOR 0
986F                ;
986F                ;GET PARAMETERS AND CHECK FOR
986F                ;ILLEGAL DEVICE. USED BY DISK
986F                ;ONLY COMMANDS.
986F                ;
986F  20 D4 E1      DPARS  JSR $E1D4       ;GET FILENAME ETC
9872  A5 BA                LDA $BA        ;IS DEVICE DISK?
9874  C9 08                CMP #$08
9876  90 05                BCC PARERR     ;NO
```

```
LOC   CODE          LINE


9878  A5 B7                LDA $B7          ;FILENAME LENGTH
987A  F0 04                BEQ FARER1       ;ZERO
987C  60                   RTS
987D  A2 09         FARERR LDX #$09          ;ILLEGAL DEVICE
987F  2C                   .BYT $2C
9880  A2 08         FARER1 LDX #$08          ;MISSING FILENAME
9882  4C 37 A4             JMP $A437        ;SEND ERROR
9985                       .END
```

OLD

Abbreviated entry: O(shift)L

Affected Basic abbreviations: None

Token: Hex $EE,$13 Decimal 238,19

Modes: Direct and program

Recommended mode: Direct only (there should be no program in memory).

Purpose: To restore a Basic program after a NEW has been performed.

Syntax: OLD

Errors: None

Use: OLD can be used if the program in memory has been wiped out using the NEW command. OLD will not work if DELETE was used to remove the whole program or if a variable has been declared since the NEW. (In most cases, a syntax error will create a variable e.g. LI instead of L(shift)I will create the variable LI and give Syntax error instead of trying to list the program).

Routine entry point: $9885

Routine operation: The first line is scanned until the end and the pointer to the next line is restored. The program is then re-chained and variable pointers are set.

```
LOC   CODE          LINE


9885                       .LIB OLD
9885  A5 2B         OLD    LDA $2B           ; FIND THE END OF
9887  18                   CLC              ; THE FIRST LINE
9889  69 04                ADC #$04
988A  85 57                STA $57          ; SET POINTER TO AFTER
988C  A5 2B                LDA $2B          ; LINE NUMBER
988E  69 00                ADC #$00
9890  85 58                STA $57+1
9892  A0 00                LDY #$00
9894  B1 57         OLD01  LDA ($57),Y       ; SEARCH LINE
9896  F0 10                BEQ OLD02        ; IF ZERO, END OF LINE
9898  A5 57                LDA $57
```

```
LOC   CODE        LINE

989A  18                CLC
989B  69 01             ADC #$01       ; INCREMENT POINTER
989D  85 57             STA $57
989F  A5 58             LDA $57+1
98A1  69 00             ADC #$00
98A3  85 58             STA $57+1
98A5  4C 94 98          JMP OLD01
98A8  A5 57       OLD02 LDA $57         ; END OF LINE
98AA  A0 00             LDY #$00       ; FOUND
98AC  18                CLC
98AD  69 01             ADC #$01
98AF  91 2B             STA ($2B),Y    ; SET NEXT LINE
98B1  C8                INY            ; POINTER
98B2  A5 58             LDA $57+1
98B4  69 00             ADC #$00
98B6  91 2B             STA ($2B),Y
98B8  4C F3 84          JMP RESVAR     ; SET VARIABLE POINTERS
98BB                    .END
```

POP

Abbreviated entry: P(shift)O

Affected Basic abbreviations: POKE – PO(shift)K

Token: Hex $EE,$14 Decimal 238,20

Modes: Direct and program

Recommended mode: Program only

Purpose: To remove the last GOSUB entry from the stack, thus leaving the subroutine without changing the execution address.

Syntax: POP

Errors: Syntax error – if POP is followed by anything but a colon or end of line marker
Return without GOSUB – if there was no GOSUB entry

Use: POP can be used in Basic programs where the user wishes to return to, say, a menu from within a Basic subroutine. If a GOTO was used without POP, after approximately 24 runs the message Out of memory will occur as the GOSUB entries will still be active. Using the POP command removes that entry and any FOR...NEXT loops active within the subroutine.

Routine entry point: $98BB

Routine operation: POP first checks for a syntax error. If there is none, the stack is scanned until the first non FOR entry is found. If it is a GOSUB, the stack pointer is set to that point and the GOSUB entry is removed. If it is not a GOSUB, the error message Return without GOSUB is displayed.

```
LOC   CODE        LINE


98BB                      .LIB POP
98BB  F0 01       POP     BEQ POPIT       ;NULL CHAR
98BD  60                  RTS            ;SYNTAX ERROR
98BE              ;
98BE  A9 FF       POPIT   LDA #$FF
98C0  85 4A               STA $4A        ;MASK OFF 'FOR'
98C2  20 8A A3            JSR $A38A      ;FIND FIRST NON 'FOR' ENTRY
98C5  C9 8D               CMP #$8D       ;GOSUB?
98C7  F0 05               BEQ DOPOP      ;YES
98C9  A2 0C               LDX #$0C
98CB  4C 37 A4            JMP $A437      ;RETURN WITHOUT GOSUB
98CE              ;
98CE  9A          DOPOP   TXS             ;MOVE POINTER TO GOSUB
98CF  68                  PLA            ;REMOVE GOSUB ENTRY
98D0  68                  PLA
98D1  68                  PLA
98D2  68                  PLA
98D3  68                  PLA
98D4  60                  RTS            ;DONE
98D5                      .END
```

PRINT

Abbreviated entry: '?'

Affected Basic abbreviations: None

Token: Hex $99   Decimal 153

Modes: Direct and program

Recommended mode: Either

Purpose: To PRINT characters to the open CMD output channel (usually value three, which is screen).

Syntax: Same as in the Basic command PRINT.

Errors: As in the Basic PRINT.

Use: This version of PRINT does exactly the same as the Basic PRINT except that a check has been made for the CTL command to be included.

Routine entry point: $98D5

Routine operation: See PRINT in Chapter 3.

```
LOC   CODE        LINE


98D5                      .LIB PRINT
98D5  20 21 AB    PRNT01  JSR $AB21       ;PRINT STRING
98D8  20 79 00    PRNT02  JSR $0079       ;GET CURRENT CHAR
98DB  F0 50       PRINTT  BEQ PRNT05      ;CARRIAGE RETURN
98DD  F0 5E       PRNT03  BEQ PRNT07      ;SEMICOLON
98DF  C9 A3               CMP #$A3       ;TAB?
```

```
LOC     CODE            LINE


98E1    F0 6B                  BEQ TAB       ;YES
98E3    C9 A6                  CMP #$A6      ;SPC?
98E5    18                     CLC
98E6    F0 66                  BEQ TAB       ;YES
98E8    C9 EE                  CMP #$EE      ;MINE?
98EA    D0 14                  BNE PRNT08    ;NO
98EC    A0 01                  LDY #$01
98EE    B1 7A                  LDA ($7A),Y   ;GET TOKEN
98F0    C9 02                  CMP #$02      ;CTL?
98F2    D0 0C                  BNE PRNT08    ;NO
98F4    20 73 00               JSR $0073
98F7    20 73 00               JSR $0073     ;GET NEXT CHAR
98FA    20 AB 88               JSR CTL       ;DO CTL
98FD    4C D8 98               JMP PRNT02
9900                    ;
9900    20 79 00        PRNT08 JSR $0079      ;GET CURRENT CHAR
9903    C9 2C                  CMP #$2C      ;','?
9905    F0 37                  BEQ PRNT09    ;YES
9907    C9 3B                  CMP #$3B      ;';'?
9909    F0 61                  BEQ TAB04     ;YES
990B    20 9E AD               JSR $AD9E     ;EVALUATE EXPRESSION
990E    24 0D                  BIT $0D       ;WHICH TYPE?
9910    30 C3                  BMI PRNT01    ;STRING
9912    20 DD BD               JSR $BDDD     ;CONVERT FAC#1 TO STRING
9915    20 87 B4               JSR $B487
9918    20 21 AB               JSR $AB21
991B    20 3B AB               JSR $AB3B
991E    D0 B8                  BNE PRNT02
9920    A9 00           PRNT04 LDA #$00
9922    9D 00 02               STA $0200,X
9925    A2 FF                  LDX #$FF
9927    A0 01                  LDY #$01
9929    A5 13                  LDA $13
992B    D0 10                  BNE PRNT07
992D    A9 0D           PRNT05 LDA #$0D       ;CARRIAGE RETURN
992F    20 47 AB               JSR $AB47
9932    24 13                  BIT $13
9934    10 05                  BPL PRNT06    ;FILE#>128 NO LF
9936    A9 0A                  LDA #$0A      ;LINE FEED
9938    20 47 AB               JSR $AB47     ;PRINT IT
993B    49 FF           PRNT06 EOR #$FF
993D    60              PRNT07 RTS
993E                    ;
993E                    ;DECIMAL TABLUATOR
993E                    ;
993E    38              PRNT09 SEC
993F    20 F0 FF               JSR $FFF0     ;GET CURSOR POS
9942    98                     TYA
9943    38                     SEC
9944    E9 0A           PRNT10 SBC #$0A       ;MINUS 10
9946    B0 FC                  BCS PRNT10
9948    49 FF                  EOR #$FF
994A    69 01                  ADC #$01
994C    D0 19                  BNE TAB01
994E                    ;
994E                    ;TAB AND SPC
994E                    ;
994E    08              TAB    PHP
994F    38                     SEC
9950    20 F0 FF               JSR $FFF0     ;GET CURSOR POSITION
9953    84 09                  STY $09       ;STORE IN TEMP
9955    20 9B B7               JSR $B79B     ;GET 1 BYTE PAR
9958    C9 29                  CMP #$29      ;')'?
995A    F0 03                  BEQ TAB10     ;YES
995C    4C 08 AF               JMP $AF08     ;SYNTAX ERROR
```

```
LOC   CODE        LINE

995F  28          TAB10  PLP            ;TAB OR SPC?
9960  90 06              BCC TAB02     ;SPC
9962  8A                 TXA           ;TAB VALUE
9963  E5 09              SBC $09       ;MINUS COLUMN POSITION
9965  90 05              BCC TAB04     ;LESS THAN
9967  AA          TAB01  TAX
9968  E8          TAB02  INX
9969  CA          TAB03  DEX
996A  D0 06              BNE TAB05
996C  20 73 00    TAB04  JSR $0073      ;GET NEXT CHAR
996F  4C DD 98           JMP PRNT03    ;BACK TO PRINT
9972  20 3B AB    TAB05  JSR $AB3B      ;OUTPUT SPACE/RIGHT
9975  D0 F2              BNE TAB03     ;ALWAYS
9977  4C 1E AB           JMP $AB1E
997A                     .END
```

PUT

Abbreviated entry: P(shift)U

Affected Basic abbreviations: None

Token: Hex $EE,$15   Decimal 238,21

Modes: Direct and program

Recommended mode: Direct

Purpose: To list a Basic program to a disk file without line numbers.

Syntax: PUT filename, d – where d is the device number (disk only).

Errors: Illegal device – if the device number specified is less than eight
Missing filename – if a null filename is specified
Device not present – if no disk drive is connected
Too many files – if ten files are already open
Disk errors – at the end, the disk error channel is read and displayed

Use: PUT is used in conjunction with GET to allow the editing of Commodore assembler source files. PUT can also be used as an alternative save method for Basic programs so that they may be run by using the EXEC command. See EXEC for an example of use.

Routine entry point: $997A

Routine operation: The filename is read along with the device number and checks are made for missing filename and illegal device number. If these are OK, the file is then opened and each line is output using the Print tokens routine to the file. At the end of each line a carriage return is set and an extra carriage return inserted at the end of the file. The file is then closed and the disk error channel is read and displayed.

```
LOC     CODE            LINE


997A                            .LIB PUT
997A    20 6F 98        PUT     JSR DPARS       ;GET FILENAME PARAMETERS
997D    20 4A 9A                JSR PUTMES      ;'WRITING..'
9980    20 F8 99                JSR PUTOPN      ;OPEN FILE
9983    20 F0 99                JSR PUTOUT      ;SET OUTPUT
9986    20 33 A5                JSR $A533       ;RE-CHAIN PROGRAM
9989    A5 2B                   LDA $2B         ;SET PROG POINTER
998B    85 5F                   STA $5F         ;TO START OF PROGRAM
998D    A5 2C                   LDA $2C
998F    85 60                   STA $60
9991    A0 00           PUT02   LDY #$00        ;END OF PROGRAM?
9993    B1 5F                   LDA ($5F),Y
9995    C8                      INY
9996    11 5F                   ORA ($5F),Y
9998    F0 12                   BEQ PUTEND      ;YES
999A    A0 04                   LDY #$04        ;POINT TO FIRST CHAR
999C    B1 5F           PUT03   LDA ($5F),Y
999E    F0 17                   BEQ PUTNL       ;END OF LINE
99A0    30 3B                   BMI PUTTK       ;PRINT TOKEN
99A2    C9 22                   CMP #$22        ;IS IT A QUOTE?
99A4    F0 29                   BEQ PUTQT       ;YES DO IT
99A6    20 D2 FF        PUT04   JSR $FFD2       ;PRINT CHAR
99A9    C8                      INY             ;SET TO NEXT
99AA    D0 F0                   BNE PUT03       ;DO NEXT (ALWAYS)
99AC                    ;
99AC    A9 0D           PUTEND  LDA #$0D        ;CARRIAGE RETURN
99AE    20 D2 FF                JSR $FFD2       ;PRINT IT
99B1    20 3A 9A                JSR PUTCLS      ;CLOSE FILE
99B4    4C 55 8A                JMP DISK01      ;DISPLAY DISK MESSAGE
99B7    A0 00           PUTNL   LDY #$00
99B9    B1 5F                   LDA ($5F),Y     ;GET LINK LO
99BB    AA                      TAX
99BC    C8                      INY
99BD    B1 5F                   LDA ($5F),Y     ;GET LINK HI
99BF    85 60                   STA $60         ;STORE AS NEXT POINTER
99C1    86 5F                   STX $5F
99C3    A9 0D                   LDA #$0D        ;CARRIAGE RETURN
99C5    20 D2 FF                JSR $FFD2       ;PRINT IT
99C8    A5 90                   LDA $90         ;STATUS
99CA    D0 E0                   BNE PUTEND      ;EXIT IF BAD
99CC    4C 91 99                JMP PUT02
99CF                    ;
99CF    20 D2 FF        PUTQT   JSR $FFD2       ;PRINT IT
99D2    C8                      INY             ;NEXT BYTE
99D3    B1 5F                   LDA ($5F),Y     ;GET BYTE
99D5    F0 E0                   BEQ PUTNL       ;END OF LINE
99D7    C9 22                   CMP #$22        ;QUOTE?
99D9    D0 F4                   BNE PUTQT       ;NO
99DB    F0 C9                   BEQ PUT04       ;OUTPUT AND DO NEXT
99DD                    ;
99DD    C9 EE           PUTTK   CMP #$EE        ;MY TOKEN?
99DF    F0 05                   BEQ PUTTK1      ;YES
99E1    20 D9 82                JSR PRIN09      ;TOKEN TO TEXT
99E4    30 03                   BMI PUTTK2      ;ALWAYS
99E6    20 BA 82        PUTTK1  JSR PRIN03      ;CONVERT TO TEXT AND PRINT
99E9    29 7F           PUTTK2  AND #$7F        ;MASK TOP BIT
99EB    A4 49                   LDY $49         ;RESTORE .Y
99ED    4C A6 99                JMP PUT04       ;SEND AND DO NEXT
99F0                    ;
99F0    A6 B8           PUTOUT  LDX $B8         ;FILE NUMBER
99F2    20 C9 FF                JSR $FFC9       ;SET OUTPUT
99F5    B0 3C                   BCS PUTOP3      ;ERROR
99F7    60                      RTS
99F8    A0 00           PUTOPN  LDY #$00
99FA    B1 BB           PUTOP1  LDA ($BB),Y     ;GET NAME BYTE
99FC    99 00 02                STA $0200,Y     ;STORE IT
```

```
LOC   CODE        LINE


99FF  C8                 INY
9A00  C4 B7              CPY $B7          ;END OF NAME?
9A02  D0 F6              BNE PUTOP1       ;NOT YET
9A04  A2 00              LDX #$00
9A06  BD 36 9A    PUTOP2 LDA PUTSW,X       ;GET BYTE
9A09  99 00 02           STA $0200,Y      ;STORE IT
9A0C  E8                 INX
9A0D  C8                 INY
9A0E  E0 04              CPX #$04         ;DONE?
9A10  D0 F4              BNE PUTOP2       ;NOT YET
9A12  A9 61              LDA #$61
9A14  85 B9              STA $B9
9A16  84 B7       PUTOP4 STY $B7           ;FILENAME LENGTH
9A18  A9 00              LDA #$00
9A1A  99 00 02           STA $0200,Y
9A1D  A0 02              LDY #$02
9A1F  85 BB              STA $BB          ;POINTER LO
9A21  84 BC              STY $BC          ;POINTER HI
9A23  A9 0E              LDA #$0E
9A25  20 A3 8A           JSR GETN1        ;GET FILE NUMBER
9A28  85 B8              STA $B8          ;FILE#
9A2A  8D 92 8D           STA EXECNO       ;FOR EXEC
9A2D  20 C0 FF           JSR $FFC0        ;OPEN
9A30  B0 01              BCS PUTOP3       ;ERROR
9A32  60                 RTS
9A33  4C F9 E0    PUTOP3 JMP $E0F9         ;OUTPUT ERROR
9A36  2C 53       PUTSW  .BYT ',S,W'
9A3A              ;
9A3A  A2 03       PUTCLS LDX #$03
9A3C  20 C9 FF           JSR $FFC9        ;OUTPUT TO SCREEN
9A3F  A2 00              LDX #$00
9A41  20 C6 FF           JSR $FFC6        ;INPUT FROM KEYBOARD
9A44  AD 92 8D           LDA EXECNO
9A47  4C C3 FF           JMP $FFC3        ;CLOSE FILE
9A4A              ;
9A4A  A9 54       PUTMES LDA #<PMESSG      ;POINTER TO MESSAGE
9A4C  A0 9A              LDY #>PMESSG
9A4E  20 1E AB           JSR $AB1E        ;PRINT MESSAGE
9A51  4C C1 F5           JMP $F5C1        ;PRINT FILENAME
9A54  57 52       PMESSG .BYT 'WRITING ',$00
9A5C  00
9A5D                     .END
```

RENUMBER

Abbreviated entry: R(shift)E

Affected Basic abbreviations: READ – RE(shift)A

Token: Hex $EE,$16 Decimal 238,22

Modes: Direct and program

Recommended mode: Direct only

Purpose: To renumber a Basic program in even line number steps. All RUNs, GOTOs, GO TOs, GOSUBs, and RUNs are renumbered if found.

Syntax: RENUMBER start,step – where start and step are values between 0 and 63999 (variables are not allowed).

Errors: Syntax error – if the syntax above is wrong
Syntax error – will occur in pass 1 if a number following any of the commands mentioned in 'Purpose' are <0 or >63999
Undefined xxxxx in old line yyyyy – if a line does not exist
Syntax error – will occur in pass 2 if the new line number is greater than 63999

Use: RENUMBER is useful for opening up program lines for the insertion of more lines or just making the program tidy after it is finished. All commands that contain line numbers will be changed so that the new line number is inserted:

RUN xxxxx
GOTO xxxxx
GO TO xxxxx
GOSUB xxxxx
THEN xxxxx
ON exp GOTO xxxxx,xxxxx.....
ON exp GOSUB xxxxx,xxxxx.....

Routine entry point: $9A5D

Routine operation: The start and step are read in and Syntax error is output if they are out of range. Pass 1 is displayed and performed. At each occurrence of a branch as above, the routine will print a '.' character. If the line does not exist, the error message Undefined xxxxx in old line yyyyy will be displayed and replaced with the number 65535 (illegal). This is done throughout the program until the end is found. Then pass 2 is displayed and the line numbers are changed to the new values. *Note:* If Syntax error is encountered in either of the passes, the renumber process will be stopped but the program will be partly renumbered and thus will not run.

```
LOC    CODE         LINE


9A5D                        .LIB RENUMBER
9A5D   20 6B A9     RENUMB  JSR $A96B         ;GET START
9A60   A5 14                LDA $14          ;LSB
9A62   8D D8 9A             STA RENSRT       ;STORE IT
9A65   A5 15                LDA $15          ;MSB
9A67   8D D9 9A             STA RENSRT+1     ;STORE IT
9A6A   20 FD AE             JSR $AEFD        ;SCAN ','
9A6D   20 6B A9             JSR $A96B        ;GET STEP
9A70   A5 14                LDA $14          ;LSB
9A72   8D DA 9A             STA RENSTP       ;STORE IT
9A75   A5 15                LDA $15          ;MSB
9A77   8D DB 9A             STA RENSTP+1     ;STORE IT
9A7A   20 8E A6             JSR $A68E        ;SET CHARGET POINTER
9A7D   20 8C 9A             JSR RENMS1       ;SEND PASS1 MESSAGE
9A80   4C 35 9B             JMP RENPS1       ;PASS 1
9A83   20 8E A6     RENU01  JSR $A68E         ;SET CHARGET POINTER
```

```
LOC   CODE         LINE


9A86  20 92 9A            JSR RENMS2     ;SEND PASS2 MESSAGE
9A89  4C E9 9A            JMP RENPS2     ;DO PASS 2 AND END
9A8C               ;
9A8C               ;TELL USER WHAT WE ARE DOING
9A8C               ;
9A8C  A9 99        RENMS1 LDA #<PS1MES    ;POINT TO
9A8E  A0 9A               LDY #>PS1MES   ;MESSAGE
9A90  D0 04               BNE RENMS3     ;SEND IT
9A92  A9 AB        RENMS2 LDA #<PS2MES    ;POINT TO
9A94  A0 9A               LDY #>PS2MES   ;MESSAGE
9A96  4C 1E AB     RENMS3 JMP $AB1E       ;OUTPUT MESSAGE
9A99               ;
9A99  2A 2A        PS1MES .BYT '**** PASS 1 ****',$0D,$00
9AA9  0D
9AAA  00
9AAB  0D           PS2MES .BYT $0D,'**** PASS 2 ****',$0D,$00
9AAC  2A 2A
9ABC  0D
9ABD  00
9ABE  0D           RENILL .BYT $0D,'UNDEFINED ',$00
9ABF  55 4E
9AC9  00
9ACA  20 49        RENIL1 .BYT ' IN OLD LINE ',$00
9AD7  00
9AD8               ;
9AD8               ;VARIABLES USED
9AD8               ;
9AD8  00 00        RENSRT .WOR 0          ;START OF RENUMBER
9ADA  00 00        RENSTP .WOR 0          ;RENUMBER STEP
9ADC  00 00        RENLNK .WOR 0          ;POINTER:START OF #
9ADE  00 00        RENLNO .WOR 0          ;POINTER:START OF LINE
9AE0  00 00        RENUST .WOR 0          ;WARM START STORE
9AE2  00           RENLEN .BYT 0          ;LENGTH:JUMP #
9AE3  00           RENLN1 .BYT 0
9AE4  00           RENTBL .BYT $00        ;DUMMY
9AE5  89                  .BYT $89       ;GOTO
9AE6  8A                  .BYT $8A       ;RUN
9AE7  8D                  .BYT $8D       ;GOSUB
9AE8  A7                  .BYT $A7       ;THEN
9AE9               RUNT    =1             ;TOKEN VALUE OF MY RUN
9AE9               ;
9AE9               ;PASS 2
9AE9               ;
9AE9  20 2A 9B     RENPS2 JSR RENU02      ;GET NEXT BYTE
9AEC  A0 00        RENPS3 LDY #$00
9AEE  B1 7A               LDA ($7A),Y    ;GET BYTE
9AF0  8D DC 9A            STA RENLNK     ;NEXT LINE LO
9AF3  C8                  INY
9AF4  B1 7A               LDA ($7A),Y    ;GET BYTE
9AF6  8D DD 9A            STA RENLNK+1   ;NEXT LINE HI
9AF9  AD D8 9A            LDA RENSRT     ;GET LINE NUMBER LO
9AFC  C8                  INY
9AFD  91 7A               STA ($7A),Y    ;STORE IT
9AFF  AD D9 9A            LDA RENSRT+1   ;HI
9B02  C8                  INY
9B03  91 7A               STA ($7A),Y    ;STORE IT
9B05  18                  CLC
9B06  AD D8 9A            LDA RENSRT     ;GET LINE# LO
9B09  6D DA 9A            ADC RENSTP     ;ADD STEP
9B0C  8D D8 9A            STA RENSRT     ;STORE IT
9B0F  AD D9 9A            LDA RENSRT+1   ;HI
9B12  6D DB 9A            ADC RENSTP+1   ;ADD STEP
9B15  8D D9 9A            STA RENSRT+1   ;STORE IT
9B18  AD DD 9A            LDA RENLNK+1   ;GET LINK HI
9B1B  F0 0A               BEQ RENUXT     ;ZERO, END OF PROG
```

```
LOC   CODE        LINE


9B1D  85 7B              STA $7B
9B1F  AD DC 9A           LDA RENLNK      ;GET LO
9B22  85 7A              STA $7A
9B24  4C EC 9A           JMP RENPS3      ;AND AGAIN
9B27  4C 74 A4    RENUXT JMP $A474        ;BACK TO 'READY'
9B2A              ;
9B2A              ;SUBROUTINE TO GET NEXT CHAR
9B2A              ; WITHOUT SCANNING PAST SPACES
9B2A              ;
9B2A  E6 7A       RENU02 INC $7A          ;BUMP LO
9B2C  D0 02              BNE RENU03
9B2E  E6 7B              INC $7B         ;BUMP HI
9B30  A0 00       RENU03 LDY #$00         ;SET INDEX
9B32  B1 7A              LDA ($7A),Y     ;GET BYTE
9B34  60                 RTS
9B35              ;
9B35              ;PASS 1
9B35              ;
9B35  20 2A 9B    RENPS1 JSR RENU02       ;GET BYTE
9B38  20 2A 9B           JSR RENU02      ;GET BYTE
9B3B  D0 03              BNE RENP01      ;NOT END OF PROG
9B3D  4C 83 9A           JMP RENU01      ;END OF PROGRAM
9B40  A5 7A       RENP01 LDA $7A          ;GET POINTER LO
9B42  8D DE 9A           STA RENLNO      ;STORE IT
9B45  A5 7B              LDA $7B         ;HI
9B47  8D DF 9A           STA RENLNO+1    ;STORE IT
9B4A  20 2A 9B           JSR RENU02      ;GET BYTE
9B4D  20 2A 9B           JSR RENU02      ;GET BYTE
9B50  20 2A 9B    RENP02 JSR RENU02       ;GET BYTE
9B53  C9 00       RENP12 CMP #$00         ;END OF LINE?
9B55  F0 DE              BEQ RENPS1      ;YES
9B57  C9 EE              CMP #$EE        ;MY TOKEN?
9B59  F0 29              BEQ RENP05      ;YES
9B5B  C9 22              CMP #$22        ;QUOTES?
9B5D  F0 1A              BEQ RENP04
9B5F  AA                 TAX
9B60  10 EE              BPL RENP02      ;NOT A TOKEN
9B62  A2 04              LDX #$04        ;LOOP TEST TOKENS
9B64  DD E4 9A    RENP03 CMP RENTBL,X     ;CHANGE IT?
9B67  F0 22              BEQ RENP06      ;YES
9B69  CA                 DEX
9B6A  D0 F8              BNE RENP03      ;DO NEXT
9B6C  C9 CB              CMP #$CB        ;IS IT 'GO'?
9B6E  D0 E0              BNE RENP02      ;NO
9B70  20 73 00           JSR $0073       ;NEXT CHARACTER
9B73  C9 A4              CMP #$A4        ;IS IT 'TO'?
9B75  D0 DC              BNE RENP12      ;NO
9B77  F0 12              BEQ RENP06      ;YES
9B79  20 2A 9B    RENP04 JSR RENU02       ;GET BYTE
9B7C  F0 B7              BEQ RENPS1      ;END OF LINE
9B7E  C9 22              CMP #$22        ;IS IT QUOTES?
9B80  F0 CE              BEQ RENP02      ;YES, DO NEXT
9B82  D0 F5              BNE RENP04      ;ALWAYS
9B84  20 2A 9B    RENP05 JSR RENU02       ;GET BYTE
9B87  C9 01              CMP #RUNT       ;RUN TOKEN?
9B89  D0 C5              BNE RENP02      ;NO
9B8B              ;
9B8B              ;ONE OF THE FIVE TOKENS HAS BEEN
9B8B              ; FOUND.
9B8B              ;
9B8B  A9 2E       RENP06 LDA #'.          ;TELL USER DOING
9B8D  20 D2 FF           JSR $FFD2       ;PRINT IT
9B90  20 73 00           JSR $0073       ;GET NEXT CHAR
9B93  90 03              BCC RENP56      ;IS A NUMBER
9B95  4C 62 9C           JMP RENU04      ;CHECK FOR ','
```

```
LOC    CODE          LINE


9B98   A5 7A         RENP56 LDA $7A          ;GET POINTER LO
9B9A   8D DC 9A             STA RENLNK      ;STORE IT
9B9D   A5 7B                LDA $7B         ;HI
9B9F   8D DD 9A             STA RENLNK+1    ;STORE IT
9BA2   A0 00                LDY #$00
9BA4   B1 7A         RENP07 LDA ($7A),Y      ;GET BYTE
9BA6   C8                   INY
9BA7   C9 30                CMP #$30        ;LESS THAN '0'?
9BA9   90 04                BCC RENP08      ;YES
9BAB   C9 3A                CMP #$3A        ;NUMERIC?
9BAD   90 F5                BCC RENP07      ;YES
9BAF   88            RENP08 DEY
9BB0   88                   DEY
9BB1   8C E2 9A             STY RENLEN      ;STORE LENGTH
9BB4   A5 7A                LDA $7A
9BB6   D0 02                BNE RENU05
9BB8   C6 7B                DEC $7B
9BBA   C6 7A         RENU05 DEC $7A
9BBC   20 73 00             JSR $0073       ;GET CHARACTER
9BBF   20 6B A9             JSR $A96B       ;GET LINE NUMBER
9BC2   20 6C 9C             JSR RENP18      ;CALCULATE NEW NUMBER
9BC5   AD DE 9A             LDA RENLNO      ;RESTORE START OF LINE
9BC8   85 7A                STA $7A         ;LO
9BCA   AD DF 9A             LDA RENLNO+1
9BCD   85 7B                STA $7B         ;HI
9BCF   20 2A 9B             JSR RENU02      ;GET LINE# LO
9BD2   85 14                STA $14         ;STORE IT
9BD4   20 2A 9B             JSR RENU02      ;HI
9BD7   85 15                STA $15         ;STORE IT
9BD9   A2 00                LDX #$00
9BDB   20 2A 9B      RENP10 JSR RENU02       ;GET BYTE
9BDE   48                   PHA
9BDF   A5 7A                LDA $7A         ;REACHED NUMBER?
9BE1   CD DC 9A             CMP RENLNK
9BE4   D0 07                BNE RENP50      ;NOT YET
9BE6   A5 7B                LDA $7B
9BE8   CD DD 9A             CMP RENLNK+1
9BEB   F0 07                BEQ RENP51      ;YES
9BED   68            RENP50 PLA
9BEE   9D 00 02             STA $0200,X     ;STORE BYTE
9BF1   E8                   INX
9BF2   D0 E7                BNE RENP10      ;ALWAYS
9BF4   68            RENP51 PLA
9BF5   A0 00                LDY #$00
9BF7   B9 00 01      RENP11 LDA $0100,Y      ;GET NEW LINE#
9BFA   F0 07                BEQ RENP13      ;END OF STRING
9BFC   9D 00 02             STA $0200,X     ;STORE IT
9BFF   C8                   INY
9C00   E8                   INX
9C01   D0 F4                BNE RENP11      ;ALWAYS
9C03   8C E3 9A      RENP13 STY RENLN1
9C06   AD E2 9A             LDA RENLEN      ;GET LENGTH
9C09   18                   CLC
9C0A   65 7A                ADC $7A         ;ADD TO POINTER
9C0C   85 7A                STA $7A         ;STORE IT
9C0E   A5 7B                LDA $7B         ;HI
9C10   69 00                ADC #$00
9C12   85 7B                STA $7B
9C14   20 2A 9B      RENP14 JSR RENU02       ;GET BYTE
9C17   9D 00 02             STA $0200,X     ;STORE IT
9C1A   F0 03                BEQ RENP15      ;END OF LINE
9C1C   E8                   INX
9C1D   D0 F5                BNE RENP14      ;ALWAYS
9C1F   8A            RENP15 TXA
9C20   18                   CLC
```

```
LOC    CODE            LINE

9C21   69 05                  ADC #$05       ;INCREASE BUFFER POINTER
9C23   85 0B                  STA $0B        ;AND STORE IT
9C25   AD 02 03               LDA $0302      ;GET WARM START LO
9C28   8D E0 9A               STA RENUST     ;STORE IT
9C2B   AD 03 03               LDA $0303      ;HI
9C2E   8D E1 9A               STA RENUST+1   ;STORE IT
9C31   A9 40                  LDA #<RENP16   ;SET WARM START
9C33   8D 02 03               STA $0302      ; VECTOR TO RETURN
9C36   A9 9C                  LDA #>RENP16   ; TO PROGRAM
9C38   8D 03 03               STA $0303      ; AFTER MAKING CHANGE
9C3B   A4 0B                  LDY $0B        ;GET BUFFER POINTER
9C3D   4C A4 A4               JMP $A4A4      ;CHANGE LINE
9C40   AD E0 9A        RENP16 LDA RENUST      ;RESTORE WARM
9C43   8D 02 03               STA $0302      ;START VECTOR
9C46   AD E1 9A               LDA RENUST+1
9C49   8D 03 03               STA $0303
9C4C   CE E3 9A               DEC RENLN1
9C4F   AD E3 9A               LDA RENLN1     ;MOVE TO END OF
9C52   18                     CLC            ;NEW LINE#
9C53   6D DC 9A               ADC RENLNK
9C56   85 7A                  STA $7A
9C58   AD DD 9A               LDA RENLNK+1
9C5B   69 00                  ADC #$00
9C5D   85 7B                  STA $7B
9C5F   20 73 00               JSR $0073      ;GET NEXT CHAR
9C62   C9 2C           RENU04 CMP #',         ;IS IT A COMMA?
9C64   F0 03                  BEQ RENP17     ;YES
9C66   4C 53 9B               JMP RENP12     ;TRY NEXT CHAR
9C69   4C 8B 9B        RENP17 JMP RENP06      ;DO NEXT LINE#
9C6C                   ;
9C6C                   ;CALCULATE NEW LINE NUMBER
9C6C                   ;
9C6C   20 8E A6        RENP18 JSR $A68E        ;SET CHARGET POINTER
9C6F   AD D8 9A               LDA RENSRT     ;SET LINE NUMBER
9C72   85 63                  STA $63
9C74   AD D9 9A               LDA RENSRT+1
9C77   85 62                  STA $62
9C79   20 2A 9B        RENP19 JSR RENU02       ;GET BYTE
9C7C   20 2A 9B               JSR RENU02     ;GET BYTE
9C7F   D0 41                  BNE RENP20     ;NOT END OF PROG
9C81   A9 9D                  LDA #$9D
9C83   20 D2 FF               JSR $FFD2
9C86   A9 20                  LDA #$20       ;FLAG ERROR
9C88   20 D2 FF               JSR $FFD2
9C8B   A9 BE                  LDA #<RENILL
9C8D   A0 9A                  LDY #>RENILL
9C8F   20 1E AB               JSR $AB1E      ;PRINT
9C92   A5 15                  LDA $15
9C94   A6 14                  LDX $14
9C96   20 CD BD               JSR $BDCD      ;PRINT NUMBER
9C99   A9 CA                  LDA #<RENIL1
9C9B   A0 9A                  LDY #>RENIL1
9C9D   20 1E AB               JSR $AB1E      ;PRINT
9CA0   AD DE 9A               LDA RENLNO
9CA3   85 FB                  STA $FB
9CA5   AD DF 9A               LDA RENLNO+1
9CA8   85 FC                  STA $FC
9CAA   A0 01                  LDY #$01
9CAC   B1 FB                  LDA ($FB),Y
9CAE   AA                     TAX
9CAF   C8                     INY
9CB0   B1 FB                  LDA ($FB),Y
9CB2   20 CD BD               JSR $BDCD      ;PRINT LINE NUMBER
9CB5   A9 0D                  LDA #$0D       ;CARRIAGE RETURN
9CB7   20 D2 FF               JSR $FFD2      ;PRINT IT
```

```
LOC   CODE        LINE


9CBA  A9 FF                 LDA #$FF      ;ILLEGAL LINE NUMBER
9CBC  85 63                 STA $63       ;65535
9CBE  85 62                 STA $62
9CC0  30 0E                 BMI RENP21    ;ALWAYS
9CC2  20 2A 9B      RENP20  JSR RENU02     ;GET BYTE
9CC5  C5 14                 CMP $14       ;SAME AS LINE#?
9CC7  D0 10                 BNE RENP22    ;NO
9CC9  20 2A 9B              JSR RENU02    ;GET BYTE
9CCC  C5 15                 CMP $15
9CCE  D0 0C                 BNE RENP23    ;NO
9CD0  A2 90         RENP21  LDX #$90
9CD2  38                    SEC
9CD3  20 49 BC              JSR $BC49     ;CONVERT LINE
9CD6  4C DF BD              JMP $BDDF     ;NUMBER TO ASCII
9CD9  20 2A 9B      RENP22  JSR RENU02     ;GET BYTE
9CDC  A5 63         RENP23  LDA $63        ;BUMP NEW LINE
9CDE  18                    CLC           ;NUMBER BY
9CDF  6D DA 9A              ADC RENSTP    ;STEP
9CE2  85 63                 STA $63
9CE4  A5 62                 LDA $62
9CE6  6D DB 9A              ADC RENSTP+1
9CE9  85 62                 STA $62
9CEB  20 2A 9B      RENP24  JSR RENU02      ;GET BYTE
9CEE  D0 FB                 BNE RENP24    ;NOT END OF LINE
9CF0  F0 97                 BEQ RENP19    ;ALWAYS
9CF2                        .END
```

REPEAT and RUN

Abbreviated entry: REPEAT RE(shift)P
RUN R(shift)U

Affected Basic abbreviations: None

Tokens: REPEAT Hex $EE,$17 Decimal 238,23
RUN Hex $EE,$01 Decimal 238,1

Modes: Direct and program

Recommended mode: Either

Purpose: REPEAT is the opening boundary of a REPEAT...UNTIL loop. RUN is the same as Basic RUN except the REPEAT stack pointer is cleared.

Syntax: REPEAT
RUN [line number]

Errors: REPEAT – Out of memory – if more than 61 nested REPEAT loops are active
RUN as in Basic RUN

Use: REPEAT...UNTIL is a very powerful looping method. For example:

```
10 REPEAT:GET A$:UNTIL A$=" "
```

will pause until the space key is pressed. The Basic version would be:

```
10 GET A$:IF A$<>“ ” THEN 10
```

The method is very simple to understand. It means REPEAT do something UNTIL done. The REPEAT...UNTIL loop does not use any of the processor stack for its storage; the RAM behind the Basic ROM is used. This enables more complicated calculations than a FOR...NEXT loop which takes up a valuable 18 bytes of the stack.

Routine entry point: REPEAT $9CF2
RUN $9D19

Routine operation: REPEAT checks for its stack being out of memory. If it is not then the command pointer and current line number are stored in the REPEAT...UNTIL stack and the stack pointer bumped (decreased) by 4. RUN just sets the REPEAT stack pointer to zero and executes the normal RUN.

```
LOC   CODE        LINE


9CF2                     .LIB REFEAT
9CF2  AD 24 9D    REFEAT LDA REFESK       ;GET STACK FOINTER
9CF5  C9 F0              CMF #240        ;ROOM ON STACK?
9CF7  D0 03              BNE REFE01      ;YES
9CF9  4C 35 A4           JMF $A435       ;'OUT OF MEMORY'
9CFC  AA          REFE01 TAX              ;STACK FOINTER
9CFD  A5 7A              LDA $7A         ;COMMAND ADDRESS LSB
9CFF  9D 00 BE           STA $BE00,X     ;STORE IT
9D02  A5 7B              LDA $7B         ;MSB
9D04  9D 01 BE           STA $BE01,X     ;STORE IT
9D07  A5 39              LDA $39         ;CURRENT LINE # LSB
9D09  9D 02 BE           STA $BE02,X     ;STORE IT
9D0C  A5 3A              LDA $3A         ;MSB
9D0E  9D 03 BE           STA $BE03,X     ;STORE IT
9D11  8A                 TXA             ;INCREASE STACK
9D12  18                 CLC             ;FOINTER BY
9D13  69 04              ADC #$04        ;4
9D15  8D 24 9D           STA REFESK
9D18  60                 RTS
9D19              ;
9D19  A9 00       RUN    LDA #$00         ;CLEAR REFEAT STACK
9D1B  8D 24 9D           STA REFESK
9D1E  20 79 00           JSR $0079       ;GET LAST CHAR
9D21  4C 71 A8           JMF $A871       ;RUN
9D24              ;
9D24  00          REFESK .BYT 0
9D25                     .END
```

SORT

Abbreviated entry: S(shift)O

Affected Basic abbreviations: None

Token: Hex $EE,$18 Decimal 238,24

Modes: Direct and program

Recommended mode: Either

Purpose: To sort a string array into alphabetically ascending order.

Syntax: SORT string array name. The string array name must be 1 or 2 bytes long, this being the characters of the name (without the $ character)

Errors: Syntax error – if no name is specified
Array not found – if the string array specified does not exist
Incorrect dimension – if the string array specified has more than one dimension
Insufficient elements – if the string array has only 1 element

Use: SORT is a bubble sort routine that will sort a string array so that all of the strings in the array can be read in alphabetically ascending order. For example:

	A$()	After SORT A
0	TEST	AFTER
1	SORT	BUBBLE
2	NAME	NAME
3	BUBBLE	READ
4	AFTER	READING
5	READING	SORT
6	READ	TEST

Routine entry point: $9D25

Routine operation: The array name is first read in and stored away in the Basic format for string arrays. The array storage area is then scanned for that array, and if not found the message Array not found is displayed. If the array is found the number of dimensions is checked, and if more than one dimension the message Incorrect dimension will be displayed. If that is OK the dimension is checked, and if it is only one value the message Insufficient elements is displayed. If all checks are OK the array is then sorted.

The method of the sort is rather complicated, and anyone wishing to know how it is done can follow the assembly listing or refer to *Library of PET subroutines* written and published by Nick Hampshire, from where the original routine was taken.

```
LOC   CODE         LINE


9D25                       .LIB SORT
9D25  20 79 00     SORT    JSR $0079          ;GET 1ST CHAR NAME
9D28  8D F0 9E             STA CA            ;STORE IT
9D2B  20 73 00             JSR $0073         ;GET 2ND CHAR
9D2E  08                   PHP
9D2F  09 80                ORA #$80          ;SET HIGH BIT
9D31  8D F1 9E             STA CB            ;STORE IT
9D34  28                   PLP               ;NULL 2ND?
9D35  F0 06                BEQ SORT00        ;YES
9D37  20 73 00             JSR $0073         ;CHARGET FOR NEXT COMMAND
```

```
LOC   CODE          LINE


9D3A  4C 42 9D             JMP SORT01
9D3D  A9 80         SORT00 LDA #$80
9D3F  8D F1 9E             STA CB
9D42                ;
9D42  A5 2F         SORT01 LDA $2F          ;SET POINTER
9D44  85 22                STA $22         ; TO ARRAY
9D46  A5 30                LDA $2F+1
9D48  85 23                STA $22+1
9D4A                ;
9D4A  A5 22         SORT02 LDA $22          ;END OF ARRAYS?
9D4C  C5 31                CMP $2F+2
9D4E  D0 0B                BNE SORT03      ;NO
9D50  A5 23                LDA $22+1
9D52  C5 32                CMP $2F+3
9D54  D0 05                BNE SORT03      ;NO
9D56  A9 00                LDA #$00        ;ARRAY NOT FOUND
9D58  4C 9B 9E             JMP SORT21
9D5B                ;
9D5B  A0 00         SORT03 LDY #$00
9D5D  B1 22                LDA ($22),Y
9D5F  CD F0 9E             CMP CA          ;NAME CORRECT?
9D62  D0 08                BNE SORT04      ;NO
9D64  C8                   INY
9D65  B1 22                LDA ($22),Y
9D67  CD F1 9E             CMP CB
9D6A  F0 1E                BEQ SORT05      ;YES
9D6C                ;
9D6C  A0 02         SORT04 LDY #$02         ;ADD LENGTH OF ENTRY
9D6E  B1 22                LDA ($22),Y     ; TO POINTER AND
9D70  8D FA 9E             STA TEMP        ; CHECK NEXT
9D73  C8                   INY
9D74  B1 22                LDA ($22),Y
9D76  8D FB 9E             STA TEMP+1
9D79  18                   CLC
9D7A  A5 22                LDA $22
9D7C  6D FA 9E             ADC TEMP
9D7F  85 22                STA $22
9D81  A5 23                LDA $22+1
9D83  6D FB 9E             ADC TEMP+1
9D86  85 23                STA $22+1
9D88  90 C0                BCC SORT02      ;ALWAYS
9D8A                ;
9D8A  A0 04         SORT05 LDY #$04
9D8C  B1 22                LDA ($22),Y     ;GET ARRAY DIMENSION
9D8E  C9 01                CMP #$01
9D90  F0 05                BEQ SORT06      ;ONLY 1 DIMENSION
9D92  A9 01                LDA #$01        ;INCORRECT DIMENSION
9D94  4C 9B 9E             JMP SORT21
9D97                ;
9D97  A0 05         SORT06 LDY #$05
9D99  B1 22                LDA ($22),Y     ;GET NUMBER OF ELEMENTS
9D9B  8D F3 9E             STA NOOFE+1
9D9E  C8                   INY
9D9F  B1 22                LDA ($22),Y
9DA1  8D F2 9E             STA NOOFE
9DA4  AD F3 9E             LDA NOOFE+1     ;ENOUGH ELEMENTS?
9DA7  D0 0C                BNE SORT07      ;YES
9DA9  AD F2 9E             LDA NOOFE
9DAC  C9 02                CMP #$02
9DAE  B0 05                BCS SORT07      ;YES
9DB0  A9 02                LDA #$02        ;TOO FEW ELEMENTS
9DB2  4C 9B 9E             JMP SORT21
9DB5                ;
9DB5  AD F2 9E      SORT07 LDA NOOFE        ;SET COUNTDOWN
9DB8  8D F4 9E             STA NOOFC       ; FOR NUMBER OF
```

```
LOC    CODE        LINE

9DBB   AD F3 9E            LDA NOOFE+1     ; MAIN SORT LOOPS
9DBE   8D F5 9E            STA NOOFC+1
9DC1                ;
9DC1   A9 00        SORT08 LDA #$00         ;MAIN LOOP OF SORT
9DC3   8D FC 9E            STA FLAGS       ;RESET SWAP FLAG,
9DC6   8D F8 9E            STA COUNT       ; AND ILOOP COUNT
9DC9   8D F9 9E            STA COUNT+1
9DCC   CE F4 9E            DEC NOOFC       ;DECREASE OLOOP COUNT
9DCF   AD F4 9E            LDA NOOFC
9DD2   C9 FF               CMP #$FF
9DD4   D0 03               BNE SORT09
9DD6   CE F5 9E            DEC NOOFC+1
9DD9                ;
9DD9   AD F5 9E     SORT09 LDA NOOFC+1      ;END OF SORT?
9DDC   D0 06               BNE SORT10      ;NO
9DDE   AD F4 9E            LDA NOOFC
9DE1   D0 01               BNE SORT10      ;NO
9DE3   60                  RTS             ;YES, DONE
9DE4                ;
9DE4   18           SORT10 CLC              ;SET $24 TO $22+7
9DE5   A5 22               LDA $22
9DE7   69 07               ADC #$07
9DE9   85 24               STA $24
9DEB   A5 23               LDA $22+1
9DED   69 00               ADC #$00
9DEF   85 25               STA $24+1
9DF1                ;
9DF1   A0 00        SORT11 LDY #$00         ;INNER LOOP
9DF3   B1 24               LDA ($24),Y     ;GET LENGTH,ADDRESS
9DF5   8D F6 9E            STA LEN1        ; OF 1ST STRING
9DF8   C8                  INY
9DF9   B1 24               LDA ($24),Y
9DFB   85 FB               STA $FB
9DFD   C8                  INY
9DFE   B1 24               LDA ($24),Y
9E00   85 FC               STA $FB+1
9E02   C8                  INY
9E03   B1 24               LDA ($24),Y     ;GET LENGTH,ADDRESS
9E05   8D F7 9E            STA LEN2        ; OF 2ND STRING
9E08   C8                  INY
9E09   B1 24               LDA ($24),Y
9E0B   85 FD               STA $FD
9E0D   C8                  INY
9E0E   B1 24               LDA ($24),Y
9E10   85 FE               STA $FD+1
9E12   AE F7 9E            LDX LEN2        ;LEN(STR2)=0?
9E15   F0 53               BEQ SORT17      ;YES, DON'T SWAP
9E17   AE F6 9E            LDX LEN1        ;LEN(STR1)=0?
9E1A   F0 28               BEQ SORT16      ;YES, SWAP THEM
9E1C   A0 00               LDY #$00
9E1E   B1 FB        SORT12 LDA ($FB),Y      ;COMPARE $FB
9E20   D1 FD               CMP ($FD),Y     ; WITH $FD
9E22   F0 05               BEQ SORT13      ;SAME
9E24   90 44               BCC SORT17      ;DIFFERENT, DON'T SWAP
9E26   4C 44 9E            JMP SORT16      ;DIFFERENT, SWAP
9E29                ;
9E29   C8           SORT13 INY              ;LENGTH=256?
9E2A   F0 3E               BEQ SORT17      ;YES, DON'T SWAP
9E2C   CC F6 9E            CPY LEN1        ;END OF STR1?
9E2F   90 04               BCC SORT14      ;NO, CHECK STR2
9E31   F0 07               BEQ SORT15      ;YES
9E33   B0 05               BCS SORT15      ;ALWAYS
9E35                ;
9E35   CC F7 9E     SORT14 CPY LEN2         ;END OF STR2?
9E38   90 E4               BCC SORT12      ;NOT YET
```

```
LOC   CODE        LINE


9E3A                     ;
9E3A  AD F6 9E    SORT15 LDA LEN1        ;LEN1=LEN2?
9E3D  CD F7 9E           CMP LEN2
9E40  F0 28              BEQ SORT17      ;YES, DON'T SWAP
9E42  90 26              BCC SORT17      ;NO, LEN1<LEN2
9E44                     ;
9E44  A0 00       SORT16 LDY #$00        ;SWAP, STR1=STR2
9E46  AD F7 9E           LDA LEN2        ; AND VICE VERSA
9E49  91 24              STA ($24),Y
9E4B  C8                 INY
9E4C  A5 FD              LDA $FD
9E4E  91 24              STA ($24),Y
9E50  C8                 INY
9E51  A5 FE              LDA $FD+1
9E53  91 24              STA ($24),Y
9E55  C8                 INY
9E56  AD F6 9E           LDA LEN1
9E59  91 24              STA ($24),Y
9E5B  C8                 INY
9E5C  A5 FB              LDA $FB
9E5E  91 24              STA ($24),Y
9E60  C8                 INY
9E61  A5 FC              LDA $FB+1
9E63  91 24              STA ($24),Y
9E65  A9 01              LDA #$01        ;FLAG SWAP
9E67  8D FC 9E           STA FLAGS
9E6A                     ;
9E6A  EE F8 9E    SORT17 INC COUNT       ;INCREMENT INNER
9E6D  D0 03              BNE SORT18      ; LOOP COUNT
9E6F  EE F9 9E           INC COUNT+1
9E72  AD F8 9E    SORT18 LDA COUNT
9E75  CD F4 9E           CMP NOOFC       ;DONE?
9E78  D0 11              BNE SORT20      ;NO
9E7A  AD F9 9E           LDA COUNT+1
9E7D  CD F5 9E           CMP NOOFC+1
9E80  D0 09              BNE SORT20      ;NO
9E82  AD FC 9E           LDA FLAGS       ;ANY SWAPS?
9E85  F0 03              BEQ SORT19      ;NO, END
9E87  4C C1 9D           JMP SORT08      ;DO NEXT LOOP
9E8A  60          SORT19 RTS             ;ALL DONE
9E8B  18          SORT20 CLC
9E8C                     ;
9E8C  A5 24              LDA $24         ;INCREASE POINTER BY 3
9E8E  69 03              ADC #$03
9E90  85 24              STA $24
9E92  A5 25              LDA $24+1
9E94  69 00              ADC #$00
9E96  85 25              STA $24+1
9E98  4C F1 9D           JMP SORT11      ;DO INNER LOOP
9E9B                     ;
9E9B  0A          SORT21 ASL A           ;SEND ERROR MESSAGE
9E9C  A8                 TAY
9E9D  B9 AD 9E           LDA POINT,Y     ;ADDRESS OF MESSAGE
9EA0  AA                 TAX
9EA1  C8                 INY
9EA2  B9 AD 9E           LDA POINT,Y
9EA5  A8                 TAY
9EA6  8A                 TXA
9EA7  20 1E AB           JSR $AB1E       ;SEND IT
9EAA  4C 62 A4           JMP $A462       ;PRINT 'IN...'
9EAD                     ;
9EAD  B3 9E       POINT  .WOR STERR1
9EAF  C4 9E              .WOR STERR2
9EB1  D9 9E              .WOR STERR3
9EB3  3F 41       STERR1 .BYT '?ARRAY NOT FOUND',$00
```

```
LOC   CODE        LINE


9EC3  00
9EC4  3F 49       STERR2 .BYT '?INCORRECT DIMENSION',$00
9ED8  00
9ED9  3F 49       STERR3 .BYT '?INSUFFICIENT ELEMENTS',$00
9EEF  00
9EF0  00          CA     .BYT 0
9EF1  00          CB     .BYT 0
9EF2  00 00       NOOFE  .WOR 0
9EF4  00 00       NOOFC  .WOR 0
9EF6  00          LEN1   .BYT 0
9EF7  00          LEN2   .BYT 0
9EF8  00 00       COUNT  .WOR 0
9EFA  00 00       TEMP   .WOR 0
9EFC  00          FLAGS  .BYT 0
9EFD                     .END
```

TRACEON and TRACEOFF

Abbreviated entry: TRACEON T(shift)R
TRACEOFF TRACEO(shift)F

Affected Basic abbreviations: None

Tokens: TRACEON Hex $EE,$19 Decimal 238,25
TRACEOFF Hex $EE,$1A Decimal 238,26

Modes: TRACEON and TRACEOFF - Direct and program

Recommended mode: TRACEON and TRACEOFF - Either

Purpose: To provide a line trace facility while the program is running for the purpose of program de-bugging.

Syntax: TRACEON
TRACEOFF

Errors: None

Use: The TRACE routine prints the current line number being executed to the current output device. If it is the screen, it will be displayed at the current cursor position.

Routine entry points: TRACEON $9EFD
TRACEOFF $9F43

Routine operation: When TRACEON is called, the line trace routine is wedged into the handle statement link. When TRACEOFF is called, the handle statement is put back into the link. The actual line trace routine first checks to see if the program is running. If not, the handle statement routine is jumped to. If the program is running, the current line is checked with the last line number displayed and if they are the same, the handle statement routine is jumped to. If it is a different line, the current line number is stored away and the line number printed thus: '[xxxxx]' and the handle statement routine is jumped to.

```
LOC     CODE            LINE


9EFD                            .LIB TRACE
9EFD    78              TRON    SEI                  ; ENABLE TRACE (TRACEON)
9EFE    A9 0A                   LDA #<TRACE
9F00    8D 08 03                STA $0308
9F03    A9 9F                   LDA #>TRACE
9F05    8D 09 03                STA $0308+1
9F08    58                      CLI
9F09    60                      RTS
9F0A                    ;
9F0A    A5 9D           TRACE   LDA $9D              ; TRACE ROUTINE
9F0C    F0 03                   BEQ TRAC01          ; ONLY IF A PROGRAM
9F0E    4C F7 82                JMP HANDLE          ; IS RUNNING
9F11    A5 39           TRAC01  LDA $39
9F13    C9 00           TRAC02  CMP #$00             ; IF SAME LINE AS
9F15    D0 09                   BNE TRAC04          ; LAST, DON'T DISPLAY
9F17    A5 3A                   LDA $39+1
9F19    C9 00           TRAC03  CMP #$00
9F1B    D0 03                   BNE TRAC04
9F1D    4C F7 82                JMP HANDLE
9F20    A5 39           TRAC04  LDA $39              ; STORE AWAY PRESENT
9F22    8D 14 9F                STA TRAC02+1        ; LINE
9F25    A5 3A                   LDA $39+1
9F27    8D 1A 9F                STA TRAC03+1
9F2A    A9 5B                   LDA #$5B
9F2C    20 D2 FF                JSR $FFD2           ; DISPLAY '['
9F2F    A6 39                   LDX $39
9F31    A5 3A                   LDA $39+1
9F33    20 CD BD                JSR $BDCD           ; DISPLAY LINE NUMBER
9F36    A9 5D                   LDA #$5D
9F38    20 D2 FF                JSR $FFD2           ; DISPLAY ']'
9F3B    A9 20                   LDA #$20
9F3D    20 D2 FF                JSR $FFD2           ; DISPLAY ' '
9F40    4C F7 82                JMP HANDLE
9F43                    ;
9F43    78              TROFF   SEI                  ; DISABLE TRACE (TRACEOFF)
9F44    A9 F7                   LDA #<HANDLE
9F46    8D 08 03                STA $0308
9F49    A9 82                   LDA #>HANDLE
9F4B    8D 09 03                STA $0308+1
9F4E    58                      CLI
9F4F    60                      RTS
9F50                            .END
```

TYPE

Abbreviated entry: T(shift)Y

Affected Basic abbreviations: None

Token: Hex $EE,$1B Decimal 238,27

Modes: Direct and program

Recommended mode: Direct

Purpose: To display a text file stored on disk to the screen.

Syntax: TYPE filename,d – where d is the device number (disk only).

Errors: Illegal device – if the device number specified is less than eight
Missing filename – if a null filename is specified
File not found – if the file does not exist
Device not present – if no disk drive is connected
Too many files – if ten files are already open
Disk errors – at the end, the disk error channel is read and displayed

Use: TYPE can be used to look at sequential files stored on disk. This can be used rather than GET if you wish to check a certain line in the file, as the file is not loaded in but directly displayed from the disk. Easyscript text files could be just as easily displayed using this routine.

Routine entry point: $9F50

Routine operation: The filename is read along with the device number and checks are made for missing filename and illegal device number. If these are OK, the file is then opened and each character is read in and displayed until the end of file or the stop key is pressed. At this point, the file is closed, the disk error channel is read and the routine exits.

```
LOC    CODE          LINE


9F50                         .LIB TYPE
9F50   20 6F 98     TYPE     JSR DPARS          ;GET FILE DETAILS
9F53   20 B7 8F              JSR GETOPN        ;OPEN FILE
9F56   20 AC 8F              JSR GETIN         ;SET INPUT
9F59   20 CF FF     TYPE2    JSR $FFCF          ;INPUT BYTE
9F5C   A6 90                 LDX $90           ;GET STATUS
9F5E   20 D2 FF              JSR $FFD2         ;PRINT BYTE
9F61   20 E1 FF              JSR $FFE1         ;STOP KEY?
9F64   F0 03                 BEQ TYPE1         ;YES
9F66   8A                    TXA
9F67   F0 F0                 BEQ TYPE2         ;NO ERROR
9F69   4C AC 99     TYPE1    JMP PUTEND         ;DONE
9F6C                         .END
```

UNTIL

Abbreviated entry: U(shift)N

Affected Basic abbreviations: None

Token: Hex $EE,$1C   Decimal 238,28

Modes: Direct and program

Recommended mode: Either

Purpose: To repeat something where the start of the Basic commands is specified by the REPEAT command until a check is true.

Syntax: UNTIL expression. The expression should be of the same format as the basic IF command.

Errors: UNTIL without REPEAT - if there was no corresponding REPEAT command

Use: UNTIL is the closing command in a REPEAT...UNTIL loop and is followed by a comparison or boolean expression. If the expression is true, the program continues running from that point. If the expression is false, the program continues from the first statement after the preceding REPEAT command.

Routine entry point: $9F6C

Routine operation: The repeat stack pointer is first checked to see if there is any active repeat. If not, UNTIL without REPEAT is displayed. If there is an active REPEAT, the expression following is checked and if the result is not zero (true) then the REPEAT...UNTIL loop is closed and exited. If the result is zero (false), the program pointers to the command following the REPEAT are set and execution starts at that point.

```
LOC   CODE        LINE

9F6C                       .LIB UNTIL
9F6C  AD 24 9D    UNTIL    LDA REPESK       ;GET STACK POINTER
9F6F  F0 39                BEQ UNTI02       ;UNTIL WITHOUT REPEAT
9F71  20 9E AD             JSR $AD9E        ;EVALUATE EXPRESSION
9F74  A5 61                LDA $61          ;GET EXPONENT
9F76  F0 0A                BEQ UNTI01       ;FALSE
9F78  AD 24 9D             LDA REPESK       ;GET STACK POINTER
9F7B  38                   SEC
9F7C  E9 04                SBC #$04         ;MINUS 4
9F7E  8D 24 9D             STA REPESK
9F81  60                   RTS
9F82  AD 24 9D    UNTI01   LDA REPESK       ;GET STACK POINTER
9F85  38                   SEC
9F86  E9 04                SBC #$04         ;MINUS 4
9F88  AA                   TAX
9F89  A5 01                LDA $01
9F8B  29 FE                AND #$FE         ;OUT BASIC
9F8D  85 01                STA $01
9F8F  BD 00 BE             LDA $BE00,X
9F92  85 7A                STA $7A          ;CHARGET POINTER LSB
9F94  BD 01 BE             LDA $BE01,X
9F97  85 7B                STA $7B          ;MSB
9F99  BD 02 BE             LDA $BE02,X
9F9C  85 39                STA $39          ;LINE# LSB
9F9E  BD 03 BE             LDA $BE03,X
9FA1  85 3A                STA $3A          ;LINE# MSB
9FA3  A5 01                LDA $01
9FA5  09 01                ORA #$01         ;IN BASIC
9FA7  85 01                STA $01
9FA9  60                   RTS
9FAA  A9 B4       UNTI02   LDA #<UNTIER
9FAC  A0 9F                LDY #>UNTIER
```

```
LOC   CODE        LINE

9FAE  20 1E AB           JSR $AB1E       ;OUTPUT ERROR
9FB1  4C 62 A4           JMP $A462
9FB4                     ;
9FB4  3F 55       UNTIER .BYT '?UNTIL WITHOUT REPEAT',$00
9FC9  00
9FCA                     .END
```

VARPTR

Abbreviated entry: V(shift)A

Affected Basic abbreviations: VAL - VAL

Token: Hex $EE,$20 Decimal 238,32

Modes: Direct and program

Recommended mode: Either

Purpose: To return the address in memory where a variable is stored.

Syntax: VARPTR (variable name). The variable name must be in ASCII characters.

Errors: Syntax error

Use: VARPTR can be used to find the address in memory of any variable be it simple or an element of an array. If the variable is a string, the value returned points to the length of the string (the following two bytes are the pointer to the actual string). For example:

> VARPTR (A$) will return the entry address of A$. To find the address of the string: DEEK(VARPTR(A$)+1)
> VARPTR (BB(12)) will return the address of the 12th element of the array BB

Routine entry point: $9FCA

Routine operation: On entry, VARPTR scans past the opening bracket and then finds the variable (or creates it if it does not exist). The closing bracket is then scanned past and the address of the variable is converted to floating point form.

```
LOC   CODE        LINE

9FCA                     .LIB VARPTR
9FCA  20 FA AE    VARPTR JSR $AEFA        ;SCAN '('
9FCD  20 8B B0           JSR $B08B       ;FIND VARIABLE
9FD0  8D E8 9F           STA VARP01      ;STORE POINTER OFF
9FD3  8C E9 9F           STY VARP01+1
```

```
LOC   CODE        LINE


9FD6  20 F7 AE              JSR $AEF7       ;SCAN PAST ')'
9FD9  A9 00                 LDA #$00        ;SET TYPE TO REAL NUMBER
9FDB  85 0D                 STA $0D
9FDD  85 0E                 STA $0E
9FDF  AE E8 9F              LDX VARP01      ;GET POINTER
9FE2  AD E9 9F              LDA VARP01+1
9FE5  4C A3 89              JMP ASSIGN      ;SEND IT
9FE8  00 00         VARP01  .WOR 0
9FEA                        .END
```

Symbol table

```
SYMBOL VALUE
 AA2R      96D7     AAERR     96B3     AALOOP    96F4     AALOP     970A
 AAMRC     971B     AASOK     96B8     ABFA      956A     ABNC      9575
 ABSC      9536     ABSLOP    9544     ADBADS    9531     ADDSUB    9501
 ADV2NC    95AB     APPEND    84D8     ARITH     8334     ARITH1    8345
 ARITH2    834F     ASARAR    9363     ASLOOP    933C     ASLOP     933E
 ASNC      934A     ASNC9     9357     ASR1R     936C     ASR2R     9378
 ASREXT    9477     ASRIR     93F2     ASRITR    943C     ASRLOP    940C
 ASRLP1    9410     ASRNC1    945C     ASRNC2    941B     ASRSOK    9388
 ASRSUB    9383     ASRTM     944C     ASRTM1    9450     ASRTM3    9469
 ASSGN     92F6     ASSIC     9300     ASSIGN    89A3     ASSR1     931D
 ASSTLO    93CE     ASSTN1    93D8     ASSTN2    93DE     ASSTN3    93E6
 AUTO      855E     AUTO01    856F     AUTO02    8573     AUTO03    857B
 AUTO04    8580     AUTO05    85A1     AUTO06    85AB     AUTOFF    8551
 AUTONO    8537     AUTOST    855C     BEXPOK    91F0     CA        9EF0
 CADDR     8189     CATL01    85C3     CATL02    85C6     CATL03    85EF
 CATL04    85F1     CATL05    860F     CATL06    8613     CATL07    8630
 CATL08    863E     CATL09    864C     CATL10    8657     CATL11    866B
 CATL12    8670     CATL13    8674     CATLOG    85B6     CB        9EF1
 CHAIN     8684     CHAIN1    86B5     CHAN01    86EA     CHAN02    86ED
 CHAN03    86F6     CHAN04    8700     CHAN05    870A     CHAN06    8716
 CHAN07    8722     CHAN08    872C     CHAN09    8742     CHAN10    8748
 CHAN11    8755     CHAN12    8764     CHAN13    876C     CHAN14    877A
 CHAN15    878F     CHAN16    87A4     CHAN17    87CB     CHAN18    87E6
 CHAN19    87F0     CHAN20    87F9     CHANGE    86B8     CHANLN    87FC
 CHANST    87FD     CHECKA    895D     CHECKB    8957     CHECKC    895B
 CHECKN    894E     CHECKS    8955     CHKOP     91BA     CHOK      9123
 CHOK1     9141     CHOK2     918D     CHOK2A    919A     CHOK3     9222
 CHRGET    0073     CHRGOT    0079     CLIST     80F1     COL       97AB
 COLD      807A     COUNT     9EF8     CRNC01    81CF     CRNC02    81DB
 CRNC03    81F1     CRNC04    81F9     CRNC05    81FC     CRNC06    81FF
 CRNC07    8201     CRNC08    8203     CRNC09    821A     CRNC10    821C
 CRNC11    8223     CRNC12    822C     CRNC13    8233     CRNC14    823D
 CRNC15    8246     CRNC16    8251     CRNC17    8253     CRNC18    8264
 CRNC19    8277     CRNC20    8279     CRNC21    8289     CRNC22    828D
 CRNCHT    81C9     CRUN01    880A     CRUN02    8818     CRUN03    881B
 CRUN04    8825     CRUN05    8830     CRUN06    883E     CRUN07    8845
 CRUN08    8849     CRUN09    8854     CRUN10    885E     CRUN11    886A
 CRUN12    8871     CRUN13    8895     CRUNCH    87FF     CTBD      8993
 CTCFLG    8994     CTCUR     8991     CTEND1    8910     CTL       88AB
 CTL01     88C1     CTL02     88CE     CTL03     88DB     CTL04     88E8
 CTL05     88F5     CTL06     88FD     CTLDEF    892C     CTLEN1    8903
 CTLEND    8906     CTSC      8992     CTXPOS    898F     CTYPOS    8990
 DEEK      8995     DELE01    89CB     DELE02    89E9     DELE03    89FA
 DELE04    8A09     DELE05    8A11     DELE06    8A26     DELE07    8A29
 DELE08    8A38     DELE09    8A40     DELE10    8A4C     DELETE    89AD
 DIMENS    8CC8     DISK      8A4D     DISK01    8A55     DISK02    8A6B
 DISK03    8A7A     DISK04    8AB8     DISK05    8ABF     DISK07    8AD4
 DOKE      8ADF     DOMAT     9249     DOMATA    92D8     DOMULT    9567
 DOPOP     98CE     DOPRNT    8321     DOSUB     955D     DPARS     986F
```

```
SYMBOL VALUE
 DUMP      8B02      DUMP01    8B0A      DUMP02    8B16      DUMP03    8B48
 DUMP04    8B81      DUMP05    8B83      DUMP06    8B90      DUMP07    8BA9
 DUMP08    8BAE      DUMP09    8BB3      DUMP10    8BB4      DUMP11    8BC0
 DUMP12    8BC5      DUMP13    8BCA      DUMP14    8BE2      DUMP15    8BE3
 DUMP16    8BEC      DUMP17    8BED      DUMP18    8BFA      DUMP19    8C06
 DUMP20    3C19      DUMP21    8C1C      DUMP22    8C56      DUMP23    8C68
 DUMP24    8C9F      DUMP25    8CA7      DUMP26    8B68      DUMTBL    8CCA
 EDVNA1    914E      EDVNA2    91A7      EDVNA3    9231      ENDCOL    9755
 ENDROW    978B      EOFMES    84AB      EXEC      8CCE      EXEC02    8D0F
 EXEC03    8D1F      EXEC04    8D36      EXEC05    8D50      EXEC06    8D59
 EXEC07    8D65      EXECER    8D90      EXECNO    8D92      EXECST    8D8E
 F1TOV1    963E      FACM      90FD      FACONT    9491      FACT      9102
 FAE1      94CD      FAEX      94ED      FAGETS    94BF      FALOOP    9480
 FANAR     94A2      FANC      94BC      FANDOK    94D2      FILENO    9861
 FIND      8D93      FIND01    8DBB      FIND02    8DC4      FIND03    8DC7
 FIND04    8DDD      FIND05    8DE5      FIND06    8DF3      FIND07    8DF9
 FIND08    8E17      FIND09    8E24      FIND10    8E56      FIND11    8E67
 FIND12    8E68      FIND13    8E8B      FIND14    8E91      FIND15    8E96
 FIND16    8EB0      FIND17    8ECA      FINDAR    9478      FINDER    8ECF
 FLAGS     9EFC      FNSTRT    001D      FOEQ      9168      FOPEN     8A91
 FUNC      8368      FUNC01    837C      FUNC02    837F      FUNC03    8394
 FUNC04    83A9      FUNC05    83B1      FUNC06    83B9      FUNC07    83BB
 FUNC08    83C6      FUNC09    83D2      FUNCTT    8B75      GADS      9671
 GAR3      92A2      GERR      8989      GET       8ED1      GETEND    8F88
 GETER     8FDE      GETFNO    8AA1      GETIN     8FAC      GETIN1    8FB4
 GETLN     8F24      GETLNO    8FDC      GETLP1    8F0F      GETLP2    8F11
 GETLP3    8F4B      GETLP4    8F80      GETMES    8F99      GETN1     8AA3
 GETN2     8AA9      GETN3     8AB4      GETN4     8AB7      GETOP1    8FB9
 GETOP2    8FC5      GETOPN    8FB7      GETSR     8FD8      GETUN     8EDB
 GETV3     91DF      GMESSG    8FA3      GV1       8965      GV2       8968
 GV3       896B      GV4       896E      HAND01    8308      HAND02    830E
 HAND03    8327      HANDLE    82F7      HIMEM     8FDF      HIMSET    8FEC
 ISNALF    B113      KEY       9014      KEY01     9024      KEY02     903C
 KEY03     904A      KEY04     904B      KEYERR    901F      KEYLO     9054
 LEN1      9EF6      LEN2      9EF7      LINK      8009      LIST01    83EB
 LIST02    83EE      LIST03    8401      LIST04    8404      LIST05    841E
 LIST06    842A      LIST07    8435      LIST08    843C      LIST09    8449
 LIST10    8457      LIST11    8470      LIST12    8494      LISTER    83E3
 LLV2      97AC      LNE       9144      LNE2      919D      LNE3      9225
 LOM01     9094      LOM02     909A      LOMEM     905D      LOMSET    906A
 MAT       9118      MERG02    97FA      MERG03    9818      MERG04    9838
 MERG05    983B      MERGE     97B0      MERGRT    9860      MERGST    986D
 MERR      9674      MMULT     90B2      MMULT1    90C2      MMULT2    90CA
 MMULT3    90CB      MMULT4    90E5      MRGMES    9862      MULT      966A
 N1        90AC      N2        90AE      NASSIG    91C7      NEAA      9794
 NOOFC     9EF4      NOOFE     9EF2      NROW      97A9      NSARR0    96A3
 NSTR1     9157      NSTR2     91B0      NSTR3     923A      NTEXP2    9185
 NTEXP3    920E      NTINT1    9161      NTINT3    9246      NUMOK     91FA
 OLD       9885      OLD01     9894      OLD02     98A8      OPDIR     8682
 OPJMP     92EC      OPJTAB    92EE      OPTYPE    910D      ORDER     9581
 PADDR     8332      PARER1    9880      PARERR    987D      PMESSG    9A54
 POINT     9EAD      POP       98BB      POPIT     98BE      POWER     80AC
 PRIN01    82A0      PRIN02    82A3      PRIN03    82BA      PRIN04    82C2
 PRIN05    82C5      PRIN06    82CD      PRIN07    82D8      PRIN08    82B4
 PRIN09    82D9      PRIN10    82E1      PRIN11    82E4      PRIN12    82EC
 PRIN13    82B7      PRINT     829E      PRINTT    98DB      PRNT01    98D5
 PRNT02    98D8      PRNT03    98DD      PRNT04    9920      PRNT05    992D
 PRNT06    993B      PRNT07    993D      PRNT08    9900      PRNT09    993E
 PRNT10    9944      PS1MES    9A99      PS2MES    9AAB      PUT       997A
 PUT02     9991      PUT03     999C      PUT04     99A6      PUTCLS    9A3A
 PUTEND    99AC      PUTMES    9A4A      PUTNL     99B7      PUTOP1    99FA
 PUTOP2    9A06      PUTOP3    9A33      PUTOP4    9A16      PUTOPN    99F8
 PUTOUT    99F0      PUTQT     99CF      PUTSW     9A36      PUTTK     99DD
 PUTTK1    99E6      PUTTK2    99E9      RENIL1    9ACA      RENILL    9ABE
 RENLEN    9AE2      RENLN1    9AE3      RENLNK    9ADC      RENLNO    9ADE
 RENMS1    9A8C      RENMS2    9A92      RENMS3    9A96      RENP01    9B40
 RENP02    9B50      RENP03    9B64      RENP04    9B79      RENP05    9B84
```

```
SYMBOL VALUE
 RENP06    9B8B     RENP07    9BA4     RENP08    9BAF     RENP10    9BDB
 RENP11    9BF7     RENP12    9B53     RENP13    9C03     RENP14    9C14
 RENP15    9C1F     RENP16    9C40     RENP17    9C69     RENP18    9C6C
 RENP19    9C79     RENP20    9CC2     RENP21    9CD0     RENP22    9CD9
 RENP23    9CDC     RENP24    9CEB     RENP50    9BED     RENP51    9BF4
 RENP56    9B98     RENPS1    9B35     RENPS2    9AE9     RENPS3    9AEC
 RENSRT    9AD8     RENSTP    9ADA     RENTBL    9AE4     RENU01    9A83
 RENU02    9B2A     RENU03    9B30     RENU04    9C62     RENU05    9BBA
 RENUMB    9A5D     RENUST    9AE0     RENUXT    9B27     REPE01    9CFC
 REPEAT    9CF2     REPESK    9D24     RESULT    90B0     RESV01    8504
 RESV02    8525     RESVAR    84F3     ROW       97AA     RUN       9D19
 RUNT      0001     SETBAS    80E5     SETKER    805D     SORT      9D25
 SORT00    9D3D     SORT01    9D42     SORT02    9D4A     SORT03    9D5B
 SORT04    9D6C     SORT05    9D8A     SORT06    9D97     SORT07    9DB5
 SORT08    9DC1     SORT09    9DD9     SORT10    9DE4     SORT11    9DF1
 SORT12    9E1E     SORT13    9E29     SORT14    9E35     SORT15    9E3A
 SORT16    9E44     SORT17    9E6A     SORT18    9E72     SORT19    9E8A
 SORT20    9E8B     SORT21    9E9B     STACK     84A9     STBAS1    80E7
 STERR1    9EB3     STERR2    9EC4     STERR3    9ED9     STKER1    8067
 STLEN     905C     SYNTE     920B     T1        9114     T2        9116
 TAB       994E     TAB01     9967     TAB02     9968     TAB03     9969
 TAB04     996C     TAB05     9972     TAB10     995F     TEMP      9EFA
 TRAC01    9F11     TRAC02    9F13     TRAC03    9F19     TRAC04    9F20
 TRACE     9F0A     TROFF     9F43     TRON      9EFD     TRPT1     95C0
 TRPT2     95B6     TRPT3     95AC     TYMISE    9152     TYPE      9F50
 TYPE1     9F69     TYPE2     9F59     UNTI01    9F82     UNTI02    9FAA
 UNTIER    9FB4     UNTIL     9F6C     V1BPT     964C     V1INT     9658
 V1REAL    925E     V2BPT     95E9     V2COLP    97AE     V2INT     95F5
 V2RA      95DD     V2TOT2    95CB     V3BPT     9621     V3INT     962D
 V3TOF1    9613     VARP01    9FE8     VARPTR    9FCA     VCOMP     898E
 VECTOR    8015     VNAME1    90F4     VNAME2    90F7     VNAME3    90FA
 VPTR1     00FB     VPTR2     00FD     VPTR3     009E     VSIZE1    9107
 VSIZE2    9109     VSIZE3    910B     VSTT1     910E     VSTT2     9110
 VSTT3     9112     VTYPE1    90F6     VTYPE2    90F9     VTYPE3    90FC
 WRST      8039     WRST01    8044     WRST02    8058

END OF ASSEMBLY
```

Index

Although it is relatively easy to learn to program the Commodore 64 in BASIC, advanced programmers need to know much more than how to use the fundamental commands. It is essential to know how BASIC works and utilises memory. The ability to add further commands is often invaluable in speeding up and simplifying a program.

This book explains all these details and sets out a unique library of routines to add extra commands to BASIC.

With this book you will not only learn far more about the Commodore 64 and its dialect of BASIC. You will also learn how to expand the scope and function of your programs; the utilities included in this book will make the writing of such programs much easier.

The Authors
Nick Hampshire is a well-known author and microcomputer expert who has specialised in Commodore computer equipment. He started the first hobby microcomputer magazine, later absorbed into *Practical Computing,* of which he was technical editor for several years. He was the co-founder of *Popular Computing Weekly* and founder and managing editor of *Commodore Computing International* magazine. He is also the author of over a dozen books on popular computing, including the very successful and widely acclaimed *PET Revealed* and *VIC Revealed.*

Richard Franklin and Carl Graham are programmers with Zifra Software Ltd and together with Nick Hampshire have written some of the software included in this book.

Also by Nick Hampshire

THE COMMODORE 64 ROMs REVEALED
0 00 383087 X

ADVANCED COMMODORE 64 GRAPHICS AND SOUND
0 00 383089 6

THE COMMODORE 64 KERNAL AND HARDWARE REVEALED
0 00 383090 X

THE COMMODORE 64 DISK DRIVE REVEALED
0 00 383091 8

COLLINS
Printed in Great Britain
0 00 383088 8

£9.95 net