1913

COMPUTER COMPANION FOR THE COMMODORE 64™

ROBERT P. HAVILAND

COMPUTER COMPANION
FOR THE
COMMODORE 64

ROBERT P. HAVILAND

TAB TAB BOOKS Inc.
BLUE RIDGE SUMMIT, PA. 17214

FIRST EDITION

FIRST PRINTING

Printed in the United States of America

Library of Congress Cataloging in Publication Data

Haviland, Robert P.
Computer companion for the Commodore 64.

Includes index.
1. Commodore 64 (Computer—Programming. 2. Basic (Computer program language) I. Title.
QA76.8.C64H38 1984 001.64'2 84-23977
ISBN 0-8306-1913-5 (pbk.)

Contents

Introduction		iv
Keywords		1
Appendix A	Operating Modes	101
Appendix B	Disk Operations	103
Appendix C	Printer Control	107
Appendix D	Master Memory Map	109
Appendix E	Graphics and the Screen Display	112
Appendix F	Sound	121
Appendix G	BASIC Program, BASIC Storage	129
Appendix H	Input and Output	132
Appendix I	Commodore 64 Codes	140
Appendix J	6501-6510 Operation Codes	150

Introduction

This little manual is intended to be a constant companion to the Commodore 64 personal computer or to any of the Commodore family that uses CBM BASIC. It is intended for use in creating programs and getting them running.

Each index tab opens to a BASIC keyword. All keywords used by CBM BASIC, version 2.0 are included, arranged alphabetically. For each keyword you are given:

—The *Name*
—The *Token* used for internal storage
—The *Class* of instruction
—The required *Form*
—The *Conditions* and results of use.

In addition examples, error messages, cautions and warnings are given where they will be helpful.

The following terminology has been used:

Function: a function is evaluated at run time and a single value is returned.

Example: SIN (x).

Command: a command causes the computer, or a computer run, to accomplish a specific operation.

Example: STOP.

Statement: a statement provides information to the computer, usually relating to program execution. Statements are sometimes grouped as declarative, conditional, and imperative, which includes commands. The family grouping used here follows the practice of the Commodore manuals.

Example: DATA

Operator: an operator, as used here, performs an arithmetic or a logical operation on a quantity or quantities to yield an arithmetic or a logical result.

Example: X AND B

Expression: An expression is a combination of statements, variables, constants, and/or operators that are reduced to

a single quantity or action at run time.

Example: IF Y=SIN(X) THEN STOP.

Run-time: run time refers to the instant at which a specified item is being worked on by the computer's central processor.

Example: Normally line 20 follows line 10 run-time.

Variables, functions and expressions is understood to be numerical unless otherwise specified. Other classes are logical, graphic, and string. Numericals can be *integer*, having no decimal point and no fractional part, or they may be *floating point*, allowing a decimal point and decimal fractional parts. Floating point numbers may include exponents, designated by E. In some cases fixed and floating point values may be intermixed; in others they must be of a specific type, or a conversion must be made. The symbol % indicates integers, and the symbol $ indicates strings. Logical items are understood by context, and graphics by appearance.

The volume includes appendices that you will find useful when you are programming. They cover the video display and graphics, sound, and input-output operations. Because the Commodore 64 version of BASIC lacks many of the commands that have been developed to control these elements on other computers, control of these must usually be accomplished through the use of PEEK and POKE instructions: the appendix material gives these in

tabular form. Both complete and partial memory maps are included.

In preparing this material, the ultimate authority has been the Commodore 64 computer itself, with a 1541 disk drive attached and an MPS-801 impact printer. The manufacturer's literature is the next level of authority, and general literature the last. However, it should be noted that both hardware and software (firmware) changes have occurred in the past and may occur again.

You should note two conventions. In the short form of keywords, the last character is entered by typing the appropriate letter with the shift key depressed; for example, the short form of PRINT# is P shifted R. This is indicated as P↑R. The second convention is the use of a bar through a graphic symbol to indicate that the symbol will be printed with the symbol and background colors interchanged or reversed.

We would appreciate a note if you find any errors which have crept in, indications of hardware or software changes by the manufacturer, or suggestions for making this series of volumes more useful.

Token 158 **Name** Absolute Value

Class Function **Alternate** A↑B

ABS

Form :ABS(quantity)

Conditions Returns the absolute value of the quantity, i.e., its magnitude less sign.

The quantity may be a number, a variable or an expression.

ABS has no meaning for string quantities.

Example ABS(−3.2)=ABS(3.2)=3.2

Note In Commodore BASIC, the argument of all functions must be enclosed in parentheses. However, in all but two cases, the (is not part of the keyword.

Token 175 **Name** Logical AND

Class Logic Operator **Alternate** A↑N

AND

Form value 1 AND value 2

Conditions Basically AND is a logical bit operator and follows these rules on a bit by bit basis:

0 AND 0=0
0 AND 1=0
1 AND 0=0
1 AND 1=1

AND returns a one at each bit location for which the values are a pair of ones; otherwise it returns 0.

Values may be a number, a variable, or an expression.

Values must be in the range −32768 to +32768.

AND has no meaning for string or graphic quantities.

Note Commonly used in statements of the form:

IF A=22 AND B=Y THEN . . .

Common in PEEK and POKE statements.

Thorough understanding of AND is necessary for bit programming of control registers.

Token 198 **Name** ASCII CODE

Class Function **Alternate** A↑S

ASC

Form ASC (string quantity)

Conditions Returns the numerical value that is the ASCII code of the first character of the string quantity.

The quantity may be a string, a string variable, or a string expression.

ASC has no meaning for numericals or for the empty string.

Example If A$ = "TEST"
PRINT ASC (A$) causes 84, the ASCII code for "T", to be printed.

Token 193 **Name** Arctangent

Class Function **Alternate** A↑T

ATN

Form ATN(value)

Conditions Returns the angle whose tangent is the value, i.e., the arctangent of the value.

The angle is in radians, in the range −PI/2 to PI/2.

The value may be a number, a variable or an expression, and may be fixed or floating point. The evaluation is floating point.

:ATN has no meaning for string or graphic quantities.

Example :ATN(1)=.785398163

Token 199 **Name** Character Code

Class String Function **Alternate** C↑H

CHR$

Form CHR$(value)

Conditions Returns the character whose ASCII code is equal to the value.

The value may be a number, a variable, or an expression, and must be in the range 0-255 (some codes do not print, and others are duplicated; see Appendix I.

The value may be fixed or floating point, but floating point values are rounded to the next smaller integer.

CHR$ has no meaning for string or graphic values.

Example CHR$(82.828)=CHR$(82)=R

Cautions The $ is automatically included in the short form. You do not have to type it.

Unfilled array elements are set to binary 0 and produce a null character.

Token 160 **Name** Close File

Class I/O Command **Alternate** CL↑O

CLOSE

Form CLOSE file number

Conditions Terminates or closes the channel to the input or output device established by the previous OPEN file number.

File number may be a number, a variable, or an expression, and must evaluate to the range 0-255.

CLOSE to a storage device (tape, disk) empties the buffer before actual termination.

CLOSE to a nonstorage device simply releases memory and terminates the channel.

See OPEN.

Example CLOSE 4

Terminates the channel to device 4, normally the printer. (Most modern printers have storage buffers.)

Note No error code is given if the channel terminated has not previously been opened.

Caution Omission of CLOSE can result in complete loss of a tape or disk file.

Token 156 **Name** Clear Variables

Class Command **Alternate** C↑L

CLR

Form CLR

Conditions Deletes the values assigned to all variables, defined functions, arrays and stacks, and sets data pointer to start.

Effectively, CLR is given at RUN, LOAD and NEW.

Disk and (tape) information and buffers are cleared, but the disk drive channel itself remains in the open state.

Caution Do not confuse CLR with the CLR/HOME key, which clears the screen.

Hint CLR resets the data pointers, but it does not actually erase the data from memory; special data recovery techniques are available.

Token 157 **Name** Change Output Mode

Class Command **Alternate** C↑M

CMD

Form CMD file number, "string"

Conditions If the specified file has been opened, CMD changes output from screen to device specified for the file.

The optional string (in quotes) is transmitted to the device. CMD can be used to send a listing or a print-list to a device.

To terminate use CLOSE; it is best to precede this with PRINT#, to ensure that the buffer is emptied.

Example

```
OPEN4,4:CMD4:LIST:PRINT#4:
CLOSE 4
```

Places a listing on the printer.

Caution Any error report will cause output to return to the screen and will CLOSE the file.

Token 154 **Name** Continue Execution

Class Command **Alternate** C↑O

CONT

Form CONT

Condition If program execution has been suspended by an END statement within the program, a STOP statement or the stop key, causes resumption of execution.

Use STOP to insert breaks in programs for debugging, examining data, or setting data values, and use CONT to resume.

Caution CONT is not functional for an error halt, or if the program has been edited during the halt, even by the deletion of a reference to a nonexistent line.

Token 190 **Name** Cosine

Class Function **Alternate** None

COS

Form COS(angle)

Conditions Returns the cosine of the angle in radians.

The angle may be a number, a variable, or an expression.

If an angle is an integer, it is converted to a floating point value.

COS has no meaning for string or graphic quantities.

Example COS(1)=.540302306

Token 131 **Name** Data Location

Class Statement **Alternate** D↑A

DATA

Form DATA value 1,value 2, . . .

Conditions Holds data to be used by READ statements.

May hold a single datum (value) or any number of values, and there may be any number of DATA statements

Values in DATA statements must be in the sequence requested by the READ statements and must correspond to the type requested (string or numeric value).

Each value must correspond to an ASCII value in the range 0-255.

To avoid error, there must be a value for each demand by the READ statements; however, excess values are ignored.

Two commas with nothing between are interpreted as a 0, or as an empty string.

String items that contain a space,

comma, colon, shifted letter (or graphic), or control character must be enclosed in quotes.

Hint Programs run faster if data statements are placed at the end of the program.

Token 150 **Name** Define Function

Class Statement **Alternate** D↑E

DEF

Form DEF FN name(variable)=formula

Conditions Establishes the formula as a callable function, for use by FN name (variable).

The name can be any legal one or two character name.

The variable is a dummy, which is replaced by the variable given by the call.

The formula can be any legal numerical expression, and does not have to include variables.

DEF FN has no meaning for string or graphic quantities.

Caution DEF FN must be executed before the function is called; the easy way to do this is to place all such definitions at the start of the program. Note that multiple executions do no harm.

Example DEF FN AB(X)=EXP(X+1) See FN.

Token 134 **Name** Dimension Array

Class Statement **Alternate** D↑I

DIM

Form DIM name (dimension1,dimension2 . . .)

Conditions Reserves memory space for the array of the given name.

The number of dimensions is the number of dimensional quantities given in parenthesis.

The number of elements in a dimension is one more than the number given for that dimension; 0 is a valid element number.

Arrays may be integer, floating point, or string, as indicated by the name.

Arrays cannot be redimensioned during program execution.

All elements of an array are set to zero at dimensioning.

If a named array that has not been dimensioned is encountered during

execution it is assigned 11 elements for each dimension; if a DIM for the name is encountered afterward, an error results.

Additional arrays may be dimensioned in a single statement by adding the construct, name (dimensions) for each added array.

Example DIM A%(1,2) produces the integer array

A%(0,0)	A%(0,1)	A%(0,2)
A%(1,0)	A%(1,1)	A%(1,2)

Hint Use of integer arrays when possible saves memory.

Cautions An out of memory error may occur at dimensioning or during execution as a result of variables encountered and stack build-up.

Practical limits for arrays are approximately

Integer	— 6,000 elements
F. Point	—18,000 elements
String	—10,000 elements

less required program space.

Error Code OUT OF MEMORY.

Token 128 **Name** End of Execution

Class Statement **Alternate** E↑N

END

Form END

Conditions When encountered during execution, END terminates execution and returns control to the keyboard.

Does not affect the variables or program.

CONT will initiate execution of any remaining program lines.

There may be any number of END statements in a program.

END is not required; execution termination and return to keyboard control occurs at the end of the last program line.

Notes STOP, END, or last line completion differ only in the report code.

The normal use of END is to separate the main program and a following subprogram.

Hint A program with many calls to a sub-program executes faster if the sub-program is at the start of the complete program.

Token 189 **Name** Exponent e

Class Function **Alternate** E↑X

EXP

Form EXP (power)

Conditions Returns the quantity e raised to the power. The power may be a number, a variable, or an expression in integer or floating point form.

USE EXP (1) to obtain e=2.71828183.

Error Code OVERFLOW if the power is greater than 88.029969191.

Token 165 **Name** Function Call

Class Function **Alternate** None

FN

Form DEF FN name(variable)=formula
FN name(value)

Conditions (a) A required part of a function definition statement.

(b) used to call the defined function.

The function definition must be executed before the call.

At call the value is substituted for the variable used in the definition.

The value may be a number, a variable, or an expression, with an integer or floating point value.

If the definition has been executed, FN is available in the direct mode.

See DEF FN.

Example If DEF FN AB(X)=EXP(X)+1,
then FN AB(2)=8.38905611.

Token 129 **Name** FOR Loop Start

Class Statement **Alternate** F↑O

FOR

Form FOR variable =initial value TO limit STEP increment

Conditions The first element of a FOR-NEXT loop.

The variable can be any floating point variable.

When FOR is encountered in execution, the variable is set to the initial value, execution continues until a NEXT or a NEXT with the same variable is encountered. The current value of the variable is then incremented by the increment or by +1 if no increment is specified. The new value is compared to the limit; if it is less, execution returns to the statement following the FOR; if it is more, execution proceeds to the statement following the NEXT.

A FOR-NEXT loop will be executed at least once.

A NEXT is required for each FOR; it may be alone, applying to the last loop

started; several NEXT variable statements may be on one line, separated by colons; or the form NEXT variable 1,variable 2 . . . may be used.

The initial, limit and increment values may be numbers, variables, or expressions, with fixed or floating point values.

A FOR-NEXT loop must be either entirely inside another (nested), or entirely outside.

A FOR-NEXT loop may be run in immediate mode if all of its elements can be placed on an 80 character logical line.

Example FOR N=1 TO 10:PRINT N:NEXT prints the numbers one through ten in a column.

Hint It is often convenient to reserve the variable names I, J, and K for quantities that must be passed from one routine to another (*global variables*), and L, M, N for variables that may be arbitrarily set and changed within a routine (*logical variables*).

Token 184 **Name** Free Memory

Class Function **Alternate** F↑R

FRE

Form FRE(any value)

Conditions Returns the number of available bytes in memory not used by BASIC.

The argument (any value) is required, but is a dummy argument. (The numbers 8 or 9 are convenient.)

If the number returned is negative, add 65536.

Caution The number returned is the number of bytes available at the execution of FRE. A program can require more memory than this, due to encountered variables, DIM statements, and the need for stacks.

Token 161 **Name** Get Character

Class Statement **Alternate** G↑E

GET

Form GET#file number, variable list
GET variable (list)

Conditions (a) Receives on character for each variable in the variable list from the device having the previously opened file number, and assigns the value of the character received to the variable. (There is no short form for this mode.)

(b) If no file number is specified, GET reads the keyboard buffer (file number=0) and assigns the buffer contents to the variable(s). Note that GET alone is usually used to get a single value typed at the keyboard.

The character received must be the correct type for the variable (numerical or string).

If no character is received, GET assigns a 0 or a null to the variable.

If there is more than one variable in the list, each following variable must be separated by a comma.

GET does not wait for a keystroke or other specified input, but it will remove a character from a full buffer, leaving room for another entry.

If file number =3 (screen), GET# reads the character at the cursor and moves the cursor one position to the right; the character at the end of a logical line is changed to a RETURN.

Compare to INPUT.

Example GET# 1,A,B,C$,D$ (gets two numerical and two string characters from the tape)

Caution Requires care when used for devices with fixed timing cycles.

Error Code SYNTAX ERROR is usually an indication of a wrong character type.

Token 203 **Name** Partial GOTO

Class Partial Statement **Alternate** None

GO

Form GO TO

Conditions Used only in the form shown, interpreted by Commodore BASIC as GOTO.

See GOTO.

Token 141 **Name** Go To Subroutine

Class Statement **Alternate** GO↑S

GOSUB

Form GOSUB line number

Conditions Transfers program control to the line number to execute a subroutine that must end with RETURN.

The line number must exist, and be given explicitly (not by a variable or an expression).

On execution of the RETURN, resumes execution at the statement following the GOSUB.

Caution While one subroutine can call another, the total number of GOSUB statements cannot exceed the number of return address held in 256 bytes of memory.

Error Code UNDEF'D STATEMENT if the designated line number does not exist.

Token 137 **Name** Unconditional Branch

Class Statement **Alternate** G↑0

GOTO

Form GOTO line number

GO TO line number

Conditions Causes an unconditional transfer of program execution to the specified line number.

The line number must exist and must be given explicitly (not by a variable or an expression).

Error Code UNDEF'D STATEMENT if the designated line number is nonexistent.

Token 139 **Name** Branch Initiation

Class Statement **Alternate** None

IF

Form

IF condition THEN action
IF condition THEN line number
IF condition GOTO line number

Conditions

Sets up a branching test, the IF-THEN construct.

The condition can include variables, strings, numbers, symbols and/or arithmetic or logical operators.

If the condition evaluates as true, i.e., non-zero, THEN the specified action or jump proceeds.

The action may be any executable BASIC command or statement; for the case of jump to a line number, only the line number is required after the THEN, but either THEN or GOTO may be used; The line number must exist.

If the condition evaluates as false, i.e., 0, all remaining elements on the line are ignored, and execution proceeds to the next available line.

The IF-THEN construct can be used in the direct mode.

Caution Multiple statements on an IF-THEN line are a major source of program errors, because they may never be implemented.

Error Code UNDEF'D STATEMENT may indicate an error in the condition or the action.

Token 133 **Name** Input from Keyboard

Class Statement **Alternate** None

INPUT

Form INPUT "prompt"; variable list

Conditions Used to secure data from the keyboard.

The prompt (with the semicolon) is optional; if used, it must conform to string rules. It will be printed on the screen, followed by a ? and the cursor. If the prompt is omitted, only the ? and the cursor appear.

The variable list may include one or more variable names, separated by commas.

The response to the screen prompt or ? should be a data list for the specified variables, correct in order, type, and number, separated by commas, and followed by the return key.

A missing data item will cause ?? to be displayed; however a null input (,, or return only) will be interpreted as 0 or as a null string.

A wrong type will cause an error message, as will an excess quantity; however, an excess will not stop execution.

INPUT can only be used within a program.

See GET, INPUT#.

Example INPUT A,B,C$,D$

Caution The only way to escape from INPUT is by holding down RUN/STOP and pressing RESTORE. The variables are not cleared by this. You can recover with GOTO line number.

The maximum length of a prompt is 38 characters.

BASIC assumes that only the last item is missing if a quantity is skipped.

Error Code REDO FROM START probably indicates a type mismatch.

?? indicates a missing quantity.

EXTRA IGNORED indicates that more than the required number of quantities were entered.

Token 132 **Name** Input from Device

Class I/O Statement **Alternate** I↑N

INPUT#

Form INPUT# file number, variable list

Conditions Accepts input from the device specified by a previous OPEN statement.

Input is accepted in the form of variable quantities, of the form, type, and order specified by the variable list, which may include one or more variables.

If the OPEN is to the keyboard (#0), it may be used to accept data from the keyboard without a prompt.

If the OPEN is to the screen (#3), it will read a logical screen line, moving the cursor to the next line; the last character is set to a CHR$(13), the RETURN key.

See INPUT, OPEN, GET.

Example INPUT# 15,A$,B$,C$,D$ will read the disk error channel if the channel is open.

Cautions A type error will halt execution.

A CHR$(13), comma, colon or semicolon not in quotes will read as the end of variable.

INPUT# is not recommended for RS-232 use; GET# is preferred.

Token 181 **Name** Integer Function

Class Function **Alternate** None

INT

Form INT(quantity)

Conditions Returns the integer value of the quantity, that is the largest integer that is equal to or less than the quantity (rounds down).

The quantity may be a number, a variable, or an expression.

INT has no meaning for string or graphic quantities.

Example

```
INT(-PI)= -4
-INT(PI)= -3
```

Token 200 **Name** Left String Slice

Class String Function **Alternate** LE↑F

LEFT$

Form LEFT$(string, length)

Conditions Returns the substring, starting from the left end of the specified string and including the number of characters specified by length.

The string may be an actual string in quotes, a string variable, or a string expression.

Length may be a number, a variable, or an expression.

If length is greater than the number of characters in the string, the entire string is returned; if it is zero or less, the null string is returned.

LEFT$ has no meaning for nonstring quantities (but these may be included in the strings).

Example LEFT$("qwerty",3)=qwe

Token 195 **Name** String Length

Class Function **Alternate** None

LEN

Form LEN(string)

Conditions Returns the length of the specified string as the number of characters, including nonprint characters and blanks.

String may be an actual string in quotes, a string variable, or a string expression.

LEN has no meaning for nonstring quantities, but these may be included in strings.

Example LEN("qwerty")=6

Token 136 **Name** Assign to Variable

Class Statement **Alternate** L↑E

LET

Form :LET variable = value
variable = value

Conditions The assignment statement, used to set the current value of the variable to the given value.

The variable may be any legal variable name (string or numerical).

The value must match the variable in type, and may be a literal string or number, a variable, or an expression.

The keyword LET is optional, as in the second form shown.

The special form

LET variable=.

is the equivalent to setting the variable to zero or the null string.

The form LET A=B=C is not allowed.

Token 155 **Name** Program List

Class Command **Alternate** L↑I

LIST

Form

LIST
LIST number1-number2

Conditions

Places a listing of the program in memory on the screen.

The program in memory is edited as it is listed. Tokens are expanded and blanks are inserted or removed.

If two line numbers separated by a dash are given, the listing starts with the first and ends with the last; if a line number does not exist, the listing starts with the next smaller or ends with the next larger line number. If a single line number followed by a dash is given, the listing starts with line number and proceeds to the end of the program; if the dash is first, the listing starts at the first line, and proceeds to the line number.

A single number with no dash will cause only that line to be displayed.

To slow the scrolling of the listing, hold

down CTRL. To stop it, use RUN/STOP key.

LIST within a program will terminate execution at the end of listing.

See CMD for printing out a listing.

Token 147 **Name** Load from Device

Class Command **Alternate** L↑O

LOAD

Form LOAD"filename",device,command

Conditions In the direct mode, LOAD initiates transfer of the name program from the specified device to memory; a program already in memory (and its variables) is lost, and all open files are closed; status messages are given.

The filename is required, except when a program is to be loaded from tape. The omission causes the first program to be loaded.

The filename may be a string in quotes, a string variable, or a string expression. A star in quotes causes the first program to be loaded; a star following a partial name causes the first file partname matching the partial name to be loaded; a ? may be placed at any letter position to indicate an unknown character.

The device may be specified by a number or by a numeric variable; if no

number is given, tape is assumed.

The command 1 means to load at the programs original place in memory; a 0 or no command causes the program to be loaded at the start of the BASIC area (normally location 2048).

A LOAD within a program can be used to find, LOAD, and RUN another program, creating a form of chain; variables from the earlier program are not erased and are available if the newer program is shorter than or the same length as the earlier program; otherwise variables may be overwritten.

Use NEW before LOAD or CLR after LOAD if old variables must be erased.

After a FOUND message in tape loads, you can press C=,CTRL, or the SPACE BAR to hurry the loading.

Example

```
LOAD
LOAD "HELLO"
LOAD "*"
LOAD "HE"+"L??",1,1
```

Caution A tape load stops the jiffy clock; see TI$

Error Code FILE NOT FOUND may indicate that the file does not exist on the device or that it is not a program file.

Token 188 **Name** Natural Logarithm

Class Function **Alternate** None

LOG

Form LOG(value)

Conditions Returns the logarithm to the base e (natural logarithm) of the value.

Value may be given as a number, a variable, or an expression, and is evaluated as a floating point quantity.

LOG has no meaning for string or graphic quantities.

Example LOG(2.5)=.916290732

Error Code ILLEGAL QUANTITY if value is zero or negative.

Token 202 **Name** Middle String Slice

Class String Function **Alternate** M↑I

MID$

Form MID$(string, start, length)

Conditions Returns the substring of string, starting from the character number given by start and of the number of characters given by length.

The string may be a literal (a string in quotes), a string variable, or a string expression.

Start and length may be numbers, variables, or expressions, and must be in the range 0-255.

If length is omitted, or there are fewer characters in the string than required by length, all rightmost characters are returned.

If start is beyond the end of the string, a null string is returned.

MID$ has no meaning for numericals, except as numbers are included in the string.

Example MID$ ("QWER"+"ASDF",3,4)
= "ERAS"

Caution The $ is automatically included in the short form. You do not have to type it.

Token 162 **Name** Prepare for New Program

Class Command **Alternate** None

NEW

Form NEW

Conditions Clears the program currently in memory in preparation for a new program.

If encountered in a program, NEW terminates execution and "erases" the program (makes it inaccessible using normal procedures).

NEW resets all pointers for variables and stacks, but does not actually erase these from memory.

Note Special programs are available to allow recovery from an accidental NEW.

Token 130 **Name** For-Loop Next

Class Statement **Alternate** N↑E

NEXT

Form NEXT variable-1,variable-2 . . .
NEXT

Conditions The required terminating element of a FOR-NEXT loop.

NEXT terminates the loop pass for the variable or variables named; if no name is given, the most recent FOR-NEXT loop is terminated.

If the loop limit has been reached or exceeded at the termination, execution proceeds to the statement following the NEXT. If it is not reached, execution returns to the statement following the FOR. In the simple form, all loops in a nest must be complete before the statement following NEXT is reached.

There must be a NEXT for each FOR; if one is not found, a search to the end of the program is made and execution halts.

See FOR,TO,STEP.

Caution After loop completion, the value of the variable will be greater than its limit by the increment or more.

Error Code READY may indicate a missing NEXT.

Token 168 **Name** Logical Not

Class Logic Operator **Alternate** N↑O

NOT

Form NOT value

Conditions Basically, the logical negating bit operator (not), obeying the rules

NOT 0=1
NOT 1=0

on a bit by bit basis.

When applied to a floating point value, NOT rounds to the next smaller integer, then proceeds as above.

The value may be given as a number, a variable, or an expression, and must be in the range −32768 to +32767.

NOT has no meaning for string or graphic quantities.

Examples

```
NOT  4= −5
NOT −5= 4
NOT −5.5= 5
```

Token 145 **Name** On Value Select

Class Statement **Alternate** None

ON

Form

ON variable GOTO line list
ON variable GOSUB line list

Conditions

A relative of the IF-THEN construct, used to select a jump/branch target line depending on the value of the variable.

If the variable evaluates to one, the target is the first line number in the line list; if it evaluates to two, the target is the second line number, and so on.

If the variable evaluates to zero or less, or to a value greater than the number of lines in the line list, the statement is ignored.

The variable may be an integer, or it may be a floating point, value, which will be rounded down during evaluation.

The target line must exist.

A string variable is not allowed, but the construct VAL (string variable) is, and

may be used in menu selection routines.

Example ON A% GOSUB 500,550,900

Note A single ON can replace a group of IF statements, and can serve as a computed GOTO.

Error Code UNDEF'D STATEMENT may indicate a nonexistent line.

Token 159 **Name** Open Channel to Device

Class I/O Statement **Alternate** O↑P

OPEN

Form OPEN file,device,command,string

Conditions Establishes a channel to a device, to allow communication by INPUT#, GET#, PRINT#.

The file may be designated by any number between 0 and 255, and this file number is required.

The device numbers are assigned as follows.

omitted	tape deck
0	keyboard
1	tape deck
2	RS-232 terminal
3	screen
4 or 5	printer
8 to 11	disk drive
4 to 127	serial bus(CR)
128 to 255	serial bus(CR/LF)

The command number is specific to each device and is required for disk; common ones are:

Tape	0 Read file 1 Write file 2 Write file with EOT
Disk	2-14 Open for data 15 Open for commands
Printer	0 Uppercase/graphics 7 Upper/lowercase

The string, in quotes, is a filename; it is required for disk file operations and optional for tape; its maximum length is 16 characters. The string may also include a file description, separated by a comma, for instance, REL for relative (the default is sequential). It may further include a mode designator, separated by comma (W=Write and R=Read, which is default). These added elements are inside the string quotation marks.

See tables of disk and printer commands in Appendices B and C.

Examples

OPEN 1,0	Read keyboard
OPEN 1,1,0	Read from tape
OPEN 1,8,15, "command"	Send the command to the disk drive.

Note Thorough understanding of OPEN is necessary for control of external devices.

Caution Failure to CLOSE OPEN files can result in the loss of data or programs on disk or tape. SYS (65511) will CLOSE all open files.

Token 176 **Name** Logical Or

Class Logic Operator **Alternate** None

OR

Form value 1 OR value 2

Conditions Basically, the logical bit operator OR, which follows these rules on a bit by bit basis.

0 OR 0= 0
1 OR 0= 1
0 OR 1= 1
1 OR 1= 1

When applied to a floating point quantity, or rounds down then proceeds as above.

Values must be between −32768 and +32767.

OR has no meaning for string or graphic quantities.

Examples

7 OR 8= 15
255 OR 15= 255
−1 OR 255= −1

Notes Common in expressions of the form:

IF A=5 OR B=1 THEN . . .

Thorough understanding of the effect of OR is necessary for bit programming of control registers.

Token 194 **Name** Peek at Address

Class Function **Alternate** P↑E

PEEK

Form PEEK (address)

Conditions Returns the byte stored in memory at the specified address.

The address must evaluate to a decimal number in the range 0 to 65535. It may be a number, a variable, or an expression.

The stored byte value is returned as a decimal integer, in the range 0-255.

Note Thorough understanding of PEEK and its companion POKE is necessary to use many of the features of machine language and of the Commodore 64 within BASIC programs.

Token 151 **Name** Poke to Address

Class Statement **Alternate** P↑O

POKE

Form POKE address,value

Conditions Used to place a byte value in memory at a specified address.

The address is entered as a decimal number and must be in the range 0 to 65535, but see the cautions below.

The value is entered as a decimal number, is converted to binary, and must be in the range 0-255.

The address and value may be given by a number, a variable, or an expression.

Note Full understanding of POKE and its companion PEEK is necessary to use many of the features of machine language and of the Commodore 64 within BASIC programs.

Caution Improper POKE locations and/or values can give errors, terminate runs, or cause loss of data or program. Consult the *Programmers Reference Guide* if you are not certain.

Token 185 **Name** Cursor Position

Class Function **Alternate** None

POS

Form POS (any value)

Conditions Returns the current position of the screen cursor.

The argument, any value, is required but is a dummy argument and is ignored; 8 is a convenient value.

The returned position is in screen columns, from 0 to 79 (values from 40 to 79 are on the second physical line of a logical screen line).

Token 153 **Name** Print to Screen

Class Statement **Alternate** ?

PRINT

Form PRINT Print list

Conditions Places the contents of the print list on the screen or as directed by CMD (see PRINT#).

A print list may include one or more of the following:

Literals, enclosed in quotes and printed as given, subject to nonprint character rules below;
Numericals, printed as given;
Variables, functions or expressions, the current value being printed.

Elements of a print list are separated by punctuation marks, which control the format as follows:

Comma—skip at least one space to the next logical column located at 0,10, . . . 70
Semicolon—no spacing.
No end punctuation—set the next print position to the start of the following line

Numerical values are preceded and followed by a space; use literal conversion if numbers must be concatenated without spaces.

The function SPC() followed by a semicolon can be used to insert 0 to 255 spaces in the print list.

The function TAB() followed by a semicolon sets the position of the following item at a specific logical screen column (0 to 255).

The nonprint characters for cursor direction, clear screen, home, and insert may be placed inside quotes, and will control the screen; overwriting is possible.

Reversed characters are obtained by using CTRL 9 inside quotes; use CTRL 0 to terminate.

The color of the characters is set by CTRL and the appropriate color inside quotes.

CMD will direct the print list to the device specified by its associated OPEN.

See the *Programmer's Reference Guide* for the use of delete and other

special characters, and the use of POKE for special effects.

Example PRINT A;B$, "try this"

Note Normally numericals must be truncated to place more than two columns on the screen.

Cautions A blank can replace format punctuation in some cases, but will stop the printing in others; the use of punctuation is recommended.

Token 152 **Name** Print to Device

Class I/O Statement **Alternate** P↑R

PRINT#

Form PRINT# file number, print list

Conditions Sends the contents of the print list to the device and file established by a previous OPEN statement.

The file must exist, and the number must be explicitly given.

The print list rules are the same as for PRINT, except for the following

The comma has same effect as a semicolon.
CR/LF (carriage return, line feed) are generated if there is no end punctuation, but are suppressed if there is.
String contents may be read as control items by the target device.

Example R$=CHR$(13)
PRINT#1,1;R$;2;R$;3;R$;4 sends spaced numbers to tape.

Note See the tables of disk and printer commands in Appendices B and C for typical uses.

Token 135 **Name** Read Data list

Class Statement **Alternate** R↑E

READ

Form READ variable list

Conditions Causes the values in one or more data lists to be entered as the current values of the variables in the variable list.

The variable list must contain a variable name immediately following the READ and may contain one or more additional variable names separated by commas.

Values are read from the data list(s) in the order set by the variable list and must agree in type.

If the length of all data lists is less than the length of a READ request, an error occurs.

If the data list(s) are longer than required by the READ requests, the extra is ignored; however this may produce an input mistake not recognized by the program.

See DATA

Caution It is usually best to have each data list contain no more than eight values, all of the same type.

Note Under some conditions, the commas may be omitted from the variable list; the practice is not recommended.

Error Code OUT OF DATA if the data list(s) are shorter than required by the READ requests.

Token 143 **Name** Remark

Class Statement **Alternate** None

REM

Form REM any text

Conditions Allows insertion of explanatory remarks into a program.

The contents of any text is ignored during program execution.

To have correct representation of graphic symbols in a REM, it is necessary to enclose them in quotes; otherwise, they will be listed as keywords. Inverse characters not in quotes will be set to normal ones.

Note 1 REM is a convenient place for short machine language routines to be called by a program.

Caution Any additional statements on the line following REM will never be executed.

Token 140 **Name** Restore Data List Pointer

Class Statement **Alternate** RE↑S

RESTORE

Form RESTORE

Conditions Resets the pointer that keeps track of the position in the data list(s) to the start of the first data list, to allow rereading of the data.

See DATA, READ.

Token 142 **Name** Return to Calling Program

Class Statement **Alternate** RE↑T

RETURN

Form RETURN

Conditions The required last statement of a subroutine.

Causes program execution to resume at the next statement after the calling GOSUB.

See GOSUB.

Caution Any statements that follow RETURN on the same program line will never be executed.

There must be a RETURN for each GOSUB that calls a different subroutine.

Token 201 **Name** Right String Slice

Class String Function **Alternate** R↑I

RIGHT$

Form RIGHT$ (string, length)

Conditions Returns the rightmost substring formed by starting at the right of string and counting back the number of characters given by length.

The string may be a literal, that is, an actual string in quotes, a string variable, or a string expression.

The length can be any integer from 0 to 255; if it is omitted the null string is returned; if it is greater than the number of characters in the string, the entire string is returned.

RIGHT$ has no meaning for numericals or graphic symbols, but these may be included in a string.

See LEFT$, MID$.

Example RIGHT$ ("QWERTY",3)="RTY"

Caution The $ is automatically included in the short form. You do not have to type it.

Token 187 **Name** Random Number

Class Function **Alternate** R↑N

RND

Form RND (value)

Conditions Returns a pseudorandom number from a sequence of 65535 such numbers.

The returned numbers are randomly distributed and are greater than zero but less than one.

The magnitude of value has no effect on RND, but the sign determines the calculation mode.

If value is positive, successive calls return successive numbers from the sequence.

If the value is negative, the return is a random selection from the sequence of values.

If the number is zero, the seed is taken from the "Jiffy clock" register "TI".

The seed is set randomly at power-up.

RND has no meaning for string or graphic quantities.

Example :INT(RND(1)*6)+INT(RND(1)*6)+2
simulates the throw of dice.

Token 138 **Name** Run Program

Class Command **Alternate** R↑U

RUN

Form

RUN
RUN line number

Conditions

Used to initiate execution of the BASIC program in memory.

RUN alone starts execution at the lowest line number in the program.

RUN followed by a number or a variable will start execution from the line given by the number or current value of the variable if the line exists; otherwise it will give an error.

RUN is preceded by an automatic CLR; use GOTO instead of RUN if the variables must be preserved.

RUN is terminated by STOP, END, execution of the last program line or by the RUN/STOP key.

Token 148 **Name** Save Program

Class Command **Alternate** S↑A

SAVE

Form

SAVE
SAVE"file-name",device,command

Conditions

If used alone, SAVE outputs the BASIC program in memory for recording on tape; the tape must be manually positioned. Overwrite of any program under the head will occur; if the record and play keys on the Commodore recorder are down, starting is automatic. The program is stored twice. There is no identification.

A filename of up to 16 characters can be used to identify files; it may be a literal, a string in quotes, a string variable, or a string expression. A filename is required with disk and optional with tape.

A device number can be included, to send the output to the required device as follows:

1	Tape
2	RS-232 device
4-5	Printer

8-11 Disk (normally 8)

(Numbers 4-127 select a serial bus device sending CR only; 128-255 sends CR/LF.)

A command may be included:

1 Reload to same memory location. (disk and tape)
2 Do not read past this point (EOT marker). (tape only)
3 Combine 1 and 2. (tape only)

If no command is given, reload is to the start of the BASIC memory area.

If the filename is preceded by @: and is the same as a filename already on the disk, the disk system replaces the previously recorded program with the current version.

See LOAD, VERIFY, and the tables of device commands in Appendix B.

Example SAVE " :TEST",8

Error Code DEVICE NOT PRESENT may indicate the misuse of command 2 or 3.

Token 180 **Name** Signum

Class Function **Alternate** S↑G

SGN

Form SGN (value)

Conditions Returns the signum of value, according to these rules:

−1 if value is negative
0 if value is zero
+1 if value is positive.

Value may be a number, a variable, or an expression.

SGN has no meaning for string or graphic quantities.

Example SGN(−3.7)=−1

Note Use in an ON-GOTO construct to select the proper routine covering negative, zero or positive cases, as for square roots of equations.

Token 191 **Name** Sine

Class Function **Alternate** S↑I

SIN

Form SIN (angle)

Conditions Returns the sine of the angle.

The angle is in radians, and may be a number, a variable, or an expression.

SIN has no meaning for string or graphic quantities.

Example SIN (0.5)=.479425539

Token 166 **Name** Insert Spaces

Class Function **Alternate** S↑P

SPC(

Form SPC (spaces)

Conditions Used in a print list to insert the specified number of spaces before the next print position.

Spaces may be a number, a variable, or an expression, and must evaluate to the range 0-255 (except 0-254 for disk).

Cautions The short form automatically includes the opening parenthesis. You do not have to type it.

The following punctuation can increase the number of spaces according to print list rules.

If SPC() causes the Commodore printer to reach the end of a line, an automatic CR/LF is given.

Token 186 **Name** Square Root

Class Function **Alternate** S↑Q

SQR

Form SQR (value)

Conditions Returns the square root of value, which must not be negative.

Value may be a number, a variable, or an expression.

SQR has no meaning for string or graphic quantities.

Example SQR (2)=1.41421356

Token None **Name** Read Error Status

Class Reserved Variable **Alternate** None

ST

Form STATUS
ST

Conditions Used as a read-variable to read the status or error report for the last I/O operation performed.

Errors are bit programmed, and any value other than 0 is an indication of some processing error.

The error indication is available in the program and direct modes.

See the device manual and the *Programmer's Reference Guide*.

Example ST=32 for tape indicates a check-sum error.

Note Do not confuse ST with the disk command channel error reports.

Token 169 **Name** Step Size

Class Statement **Alternate** ST↑E

STEP

Form FOR variable=initial TO limit STEP increment

Conditions The optional final element of the FOR part of a FOR-NEXT loop.

Establishes the increment between one pass through the loop and the next.

The STEP increment can be omitted, giving a default increment of +1.

The increment may be a number, a variable, or an expression, in fixed or floating point form (noninteger increments are allowed).

The value of increment is set at the start of loop execution and is unchangeable in BASIC. See FOR, TO, NEXT.

Caution An increment of 0 creates an infinite loop when executed.

Token 144 **Name** Stop Execution

Class Statement **Alternate** S↑T

STOP

Form STOP

Conditions Halts program execution and returns control to the keyboard; however, files remain open and variables are not cleared.

CONT causes execution to resume with the following statement, except if the program has been edited in any way. (However, variables can be examined or changed.)

STOP is identical to END except in the report generated.

STOP is identical to pressing the RUN/STOP key if that execution is halted at the end of the current statement.

There can be any number of STOP statements in a program.

Notes Use STOP to set breakpoints in programs.

Use GOTO if the program is edited after a STOP.

Caution Files remain OPEN after a STOP.

Error Code BREAK IN line number may indicate STOP or a keyboard RUN/STOP.

Token 196 **Name** Convert to String

Class String Function **Alternate** ST↑R

STR$

Form STR$ (value)

Conditions Converts a numeric value to its corresponding string.

After conversion, the element must be manipulated by string rules.

The value may be a number, a variable whose current value is converted, or an expression that is evaluated and the result converted.

A converted number is preceded by a space.

Example STR$(2*6)="12"

Cautions The $ is automatically included in the short form. You do not have to type it.

Token 158 **Name** Call Machine Language Routine

Class Statement **Alternate** S↑Y

SYS

Form SYS location

Conditions Transfers processing to the machine language routine residing in memory at the specified location.

Location may be a number, a variable or an expression, and must be in the range 0-65535.

Machine language processing continues until code 60h (RTS, Return from Subroutine) is encountered, subject to routine return rules. Program processing then resumes with the next BASIC statement.

See USR and the system memory maps in Appendices D, E, and F.

Notes SYS is called SYS(tem) because it is a convenient way to use the many CBM routines built into the computer.

SYS may be used to call special routines anywhere in memory.

A good understanding of machine language is necessary to full use of the computer.

Caution Improper locations or code errors can suspend execution and/or erase programs and memory contents.

Token 163 **Name** Tabular Set

Class Function **Alternate** T↑A

TAB(

Form TAB(column)

Conditions Places the print position at the specified screen or device column.

TAB(functions only following PRINT or in a print list.

If the current print position is beyond the specified column, the print position moves to the next line and the specified column; the first column is 0 and the last screen column is 39. (This may be changed by POKE (53270).) The last printer column is 79.

Column may be a number, a variable, or an expression, in the range 0-255; values larger than the last position cause additional lines to be skipped.

When TAB is used in a print list, the following punctuation change the print position according to print list rules.

Example PRINT TAB(19);"X" Places an X on the center line.

Caution The (is already a part of the short form. You do not need to type it when you use the short form.

Error Code SYNTAX ERROR may indicate an excess (with the short form) or that a PRINT is necessary.

Token 192 **Name** Tangent

Class Function **Alternate** None

TAN

Form TAN (angle)

Conditions Returns the tangent of the angle, which is entered in radians.

Angle may be a number, a variable, or an expression, and must not be equal to zero.

TAN has no meaning for string or graphic quantities.

Example TAN (10)=.648360828

Error Code DIVISION BY ZERO if the angle is PI/2 or precise odd multiples thereof.

Token 167 **Name** Branch Terminator

Class Statement **Alternate** T↑H

THEN

Form

IF condition THEN action
IF condition THEN line number

Conditions

Is the required second part of the IF-THEN construct.

Establishes the action to be taken if the condition is true, that is, non-zero.

The action may be any executable BASIC statement; for the case of jump to a line number, the GOTO is optional (either THEN or GOTO may be used), but the line number must exist.

The IF-THEN construct can be used in the direct mode.

Caution

If the condition is not satisfied, execution proceeds to the next program line, not just to the next statement; additional statements following the IF-THEN perform useful functions, but are a frequent source of errors.

Token None **Name** Timer

Class Reserved Variable **Alternate** None

TI

Form TI

Conditions Reflects the value of an interrupt driven timer called the *jiffy clock* in units of 1/60 second.

The timer reading is incremented once for each screen scan, and has a maximum reading of 5,184,000 or 24 hours. The timer is initially set at power-up, or as commanded with TI$.

The timer does not run during tape operations.

The clock rate is crystal controlled, but there will be small variations in rate from unit to unit and with varying temperatures.

See TI$.

Notes In addition to the jiffy clock, there are a total of four interval timers and two 24 hour clocks available.

Caution The jiffy timer is halted during tape operations.

Token None **Name** Time String

Class Reserved Variable **Alternate** None

TI$

Form TI$="hhmmss"
TI$

Conditions In the first form above, TI$ sets the jiffy clock to the number of one-sixtieth second intervals that are in the period from 0 to the specified hour, minute and second.

When used as a read variable, TI$ returns the jiffy clock reading in the hhmmss format.

See TI.

Note In addition to the jiffy clock, there are a total of four interval timers and two 24 hour clock alarms available.

Cautions The jiffy clock stops during tape I/O operation.

Token 164 **Name** Loop To Value

Class Statement **Alternate** None

TO

Form FOR variable=initial TO limit STEP increment

Conditions The required second element of a FOR-NEXT loop.

Establishes the limiting value of the variable, against which the current value is tested when the corresponding NEXT is encountered.

See FOR, NEXT, STEP

Notes TO also appears to be in GO TO, but this is simply an accepted form of GOTO.

Token 183 **Name** User Routine

Class Function **Alternate** U↑S

USR

Form USR (value)

Conditions Calls the user's machine language routine whose starting address is pointed to by memory locations 785-786, which must be previously set.

Value is stored in the floating point accumulator starting at location 97, and can be used as an argument or parameter of the routine.

The routine can return one or more quantities by using the accumulator or other storage locations.

The routine must end with RTS (code 60h—Return from Subroutine)

See SYS.

Note Good understanding of machine code is needed for full use of USR and SYS; see the *Programmer's Reference Guide*.

Token 197 **Name** String to Value Convert

Class Function **Alternate** V↑A

VAL

Form VAL (string quantity)

Conditions Returns as a numeric the value of the digits in a string quantity.

If the first character of the string is not +, −, a decimal point or a digit, VAL returns 0.

With a numeric first character, succeeding digits (including E for exponent) are returned, until the first non-numeric character is encountered; only one decimal point is returned.

See STR$.

VAL("1.2E3 alpha")=1200

Token 149 **Name** Verify Program

Class Command **Alternate** V↓E

VERIFY

Form VERIFY "file name", device
VERIFY

Conditions Checks the program "filename" on tape or disk against the program in memory, providing a match or doesn't match indication.

If name and device are omitted, the test is of the first program on tape; if only device is omitted, the named program on tape is tested.

The name can be a string in quotes, a string variable, or a string expression.

Example VERIFY "STO"+"*",8

checks the first program on disk whose name starts with STO.

Notes While SAVE error rates are low, it is good practice to VERIFY all long or important programs.

To position the tape for a new program,

enter the command to VERIFY the last program recorded; At the error message, the tape is safely at a clear area.

Token 146 **Name** Wait an Interval

Class Statement **Alternate** W↑A

WAIT

Form WAIT location, mask1, mask2

Conditions Suspends program execution until the bit pattern stored at the specified memory location changes to the value specified by the masks, which must be in the range 0-255.

Mask1 specifies the bits to be tested, by being ANDed with the location contents.

The optional mask2 specifies the bit pattern to be accepted as termination, by XOR with the location contents (a bit to be tested for 0 should be set to 1 in mask2); if mask2 is omitted, the bits accepted by mask1 are checked for ones.

Example TI$="000000":WAIT 161,1,0 produces a wait of 256 jiffies, or 4.27 seconds.

Note WAIT is normally used in I/O timing, but can also be used for interrupt servicing, process control, and so on. In simple

applications, use TI or TI$ instead.

Caution An infinite wait state can result from improper mask values.

Appendix A
Operating Modes

General

Maximum of 40 characters in a screen line.
Maximum of 80 characters in an executable (logical) line.
Allows multiple statements in a line, separated by colons.
Character set 1 is the uppercase; shift gives the right front symbols; C= gives the left front symbols.
Character set 2 is the lowercase; shift gives the uppercase, C= gives the left front symbols.
Shift C= switches between character sets.
"Cold" start=SYS 64738=clear all.

Screen Control

Bled screen=POKE 53280,6 (blue border and screen).

B/W screen=CTRL 2 followed by POKE 53281,0.
Border color=POKE53280 C; C = color.
Screen Color=POKE 53281,C
Blank screen=POKE 53265,(PEEK (53265) AND 239)
Normal screen=POKE 53265,(PEEK (53265) OR 16)

Screen Format

Columns	38=POKE 53270,PEEK (53270) AND 247 40=POKE 53270,PEEK (53270) OR 8
Rows	24=POKE 53265,PEEK (53265) AND 247 25=POKE 53265,PEEK (53265) OR 8

Appendix B
Disk Operations

Basic Disk Operations

LOAD"name",8,blank/0/1	Program load
LOAD"$",8	Directory load
LOAD "part-name*",8	Short form load (will load first program beginning with part-name)
SAVE"name",8,blank/0/1	Program save
SAVE"@:name",8,blank/0/1	Program save, replace old version
VERIFY"name",8	Program verify

Extended Disk Operations

OPEN file number,8, channel,"text"	Basic form of OPEN
OPEN15,8,15	Command OPEN
PRINT#15,"command"	Transmit command
PRINT#15,"N0:name,ID"	Format disk

PRINT#15,"N0:disk-name"	Clear directory
PRINT#15,"C0:newfile=0 :oldfile"	Copy file
PRINT#15,"C0:newfile=0 :oldfile1,0:.."	Combine files
PRINT#15,"R0 :newname=oldname"	Rename file
PRINT#15,"S0:name"	Scratch file
PRINT#15,"I"	Return disk to its startup condition
PRINT#15,"V"	Rewrite and reorganize disk

Sequential Files

(Direction is R=Read, W=Write)

OPEN file number,8,channel,"0:name,S,direction"

(Note add @ before the 0 to replace old file.)

PRINT# file number, data list	write
INPUT# file number, variable list	read
GET# file number,variable list	read by character(s)

Relative Files

(Check error channel after each operation.)

OPEN 15,8,15	for commands
OPEN file number,8,	

channel,"name,L" +CHR$(length)	originate a file
PRINT# file number, data list	write to a file
OPEN file number,8, channel,"name"	reopen a file
PRINT#15,"P"CHR$ (channel)CHR$(lo-address)CHR$(hi-address) CHR$(position)	position file pointer
PRINT# file number part data list	write to field
INPUT# file number, data list	read all or part
GET# file number,data list	read all or part by character(s)

Error Channel

INPUT#15 ERR,ERR$,T,B	get status message

(Note: IF ERR<20 OR ERR>50 THEN things are okay)

File Closing

CLOSE file number	close file first
CLOSE 15	close error channel last

WARNING FAILURE TO CLOSE OPEN FILES MAY CAUSE LOSS OF DATA. The V command can destroy random files.

NOTES The / should be read as or.
The 1 causes a load to the same

memory locations from which it came; blank or 0 causes a load to the start of the area available for BASIC programs.
Device 8 may be changed to 9-11.
With multidrives, 0 may be 1.
The two character ID is assigned to all blocks of the disk.
Any command may be used with OPEN.
Channels 0 and 1 are reserved;2-14 are assignable;15 is command.
See instruction book for random and user files, and for DOS routine access.

Appendix C
Printer Control

OPEN filenumber,4,option

OPTION= 0 print as received
= 6 space lines
= 7 upper/lower case
= 8 upper/graphics case
= 10 reset

CMD filenumber

Printer is open and waiting for further instructions

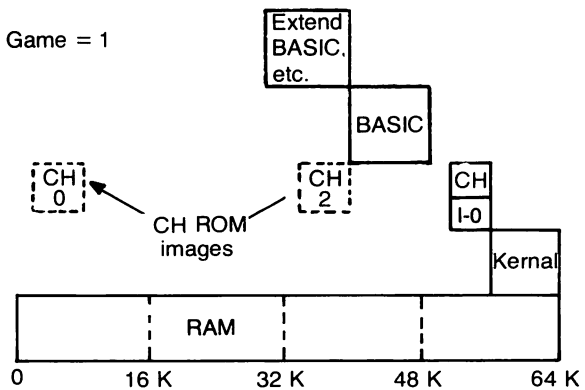
PRINT# filenumber, print list

Print list commands

CHR$ (8) Graphics mode (9 lines/in)
CHR$ (10) Line feed
CHR$ (13) Carriage return

CHR$ (14)	Double width print
CHR$ (15)	Standard print
CHR$ (16)	Tab set (2 digits follow)
CHR$ (17)	Lowercase characters (uppercase with shift)
CHR$ (18)	Inverse characters
CHR$ (26)	Repeat graphics
CHR$ (27)	Dot address (9 bits follow)
CHR$(145)	Uppercase characters (graphics with shift)
CHR$(146)	Inverse off

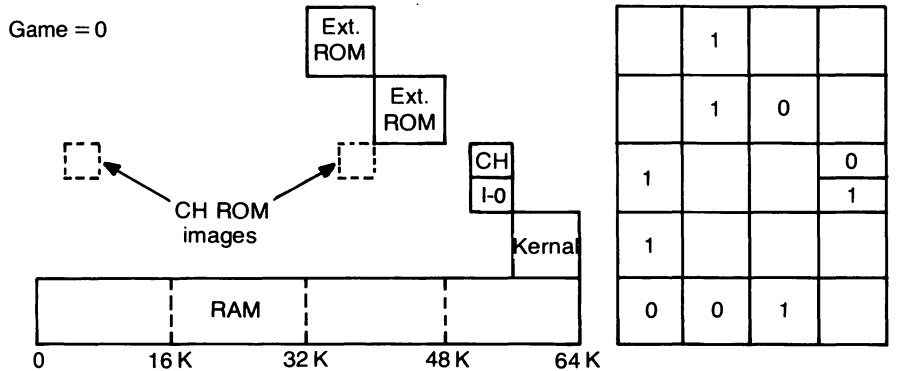
Appendix D
Master Memory Map



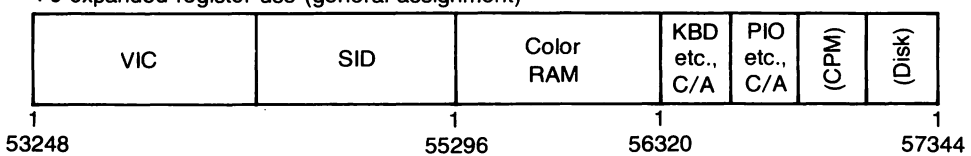
Control bits

Hi RAM	Lo RAM	Ex ROM	Charen
1		0	
1	1		
1			0
			1
1			
0	0	1	

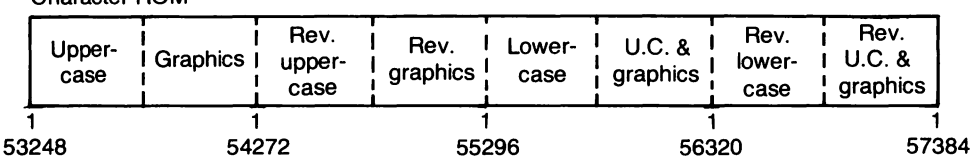
Note: CH = character ROM
I-0: see below
Table blanks = don't care



I-0 expanded register use (general assignment)



Character ROM



Appendix E

Graphics and the Screen Display

SCREEN COLORS

Code	Color	Default Sprite #
0	Black	—
1	White	0
2	Red	1
3	Cyan	2
4	Purple	3
5	Green	4
6	Blue	5
7	Yellow	6

The following are not available in the multicolor mode.

8	Orange	—
9	Brown	—
10	Lt. Red	—

11	Gray 1	—
12	Gray 2	7
13	Lt. Green	—
14	Lt. Blue	—
15	Gray 3	—

Screen Color data is stored at 55296+X+40*Y
where X=Horizontal column location, 0-39
Y=Vertical row location, 0-24

The normal screen data is stored at 1024+X+40*Y

Select 16K Memory Block

```
POKE 56576,(PEEK(56576)
AND 252)OR A
A=0 gives location 49152,
A=3=normal
```

Select 1K Graphic location in block

```
POKE 53272,(PEEK(53272)
AND 15) OR A
A=0,16,32,. . .240
```

Color Memory=55296-56295 (location fixed)

Select 2056 byte Character Set in Block

```
POKE 53272,(PEEK(53272) AND 240) OR A
A=0,2,4,. . .14 (4 is normal)
Character address=(Screen Code)*8+
(Character Set #)*2048+(Bank)*16
```

Character ROM

```
In    POKE 1,PEEK(1) AND 251
Out   POKE 1,PEEK(1) OR 4
```

MultiColor Mode

```
On    POKE 53270,PEEK(53270) OR 16
Off   POKE 53270,PEEK(53270) AND 239
```

Extended Background Color Mode

```
On    POKE 53265,PEEK(53265) OR 64
Off   POKE 53265,PEEK(53265) AND 191
```

Bit Map Mode

On	POKE 53265,PEEK(53265) OR 32
Off	POKE 53265,PEEK(53265) AND 223

Screen Enable

On	POKE 53265,PEEK(53265) AND 239
Off	POKE 53265,PEEK(53265) OR 16

Screen Size

38 Columns	POKE 53270,PEEK(53270) AND 247
40 Columns	POKE 53270,PEEK(53270) OR 8
24 Rows	POKE 53265,PEEK(53265) AND 247
25 Rows	POKE 53265,PEEK(53265) OR 8

Smooth Scroll by 0-7 X or Y Pixels

Y(set 24 rows)	POKE 53265,PEEK(53265) AND 248+Y
X(set 38 columns)	POKE 53270,PEEK(53270) AND 248+X

SPRITE SUMMARY

LET V=Start of Video Chip =53248
M=Start of sprite memory storage
=832 (for a maximum of 3 sprites)
= 12288 (typical)
SN=sprite # (0-7) 0=highest priority
P=sprite pixel byte (0-62)
C=Color (0-15) Black=0

Then to

Store	POKE M+64*SN+P,Byte-value
Point to	POKE 2040+SN,M/64+SN (normal screen)
Turn on	POKE V+21,PEEK(V+21)OR(2↑SN)
Turn off	POKE V+21,PEEK(V+21)AND (255-2↑SN)
Color	POKE V+39+SN,C
Place	
Main X	POKE V+2*SN,X X=0-255
Left X	POKE V+16, PEEK(V+16)AND (255-2↑SN)
Right X	POKE V+16,PEEK(V+16)OR(2↑SN)
Main Y	POKE V+1+2*SN,Y Y=0-255
X-Expand	POKE V+29,PEEK(V+29)OR(2↑SN)
Normal	POKE V+29,PEEK(V+29)AND (255-2↑SN)
Y-Expand	POKE V+23,PEEK(V+23)OR(2↑SN)
Normal	POKE V+23,PEEK(V+23)AND (255-2↑SN)
Multicolor	POKE V+28,PEEK(V+28)OR(2↑SN)

Color-0	POKE V+37,C
Color-1	POKE V+38,C
Normal	POKE V+28,PEEK(V+280)AND (255-2↑SN)

Read collisions (cleared by reading)

Sprite-sprite	PEEK(V+30)AND(2↑SN)
Sprite-data	PEEK(V+31)AND(2↑SN)

Read Light-Pen Position

PEEK (53267)	X-Position to two dot resolution
PEEK (53268)	Y-Position to one line resolution

Note: average of three readings recommended.

Read Raster-Line Position

PEEK (53266)+ 256* (PEEK (53265) AND 128

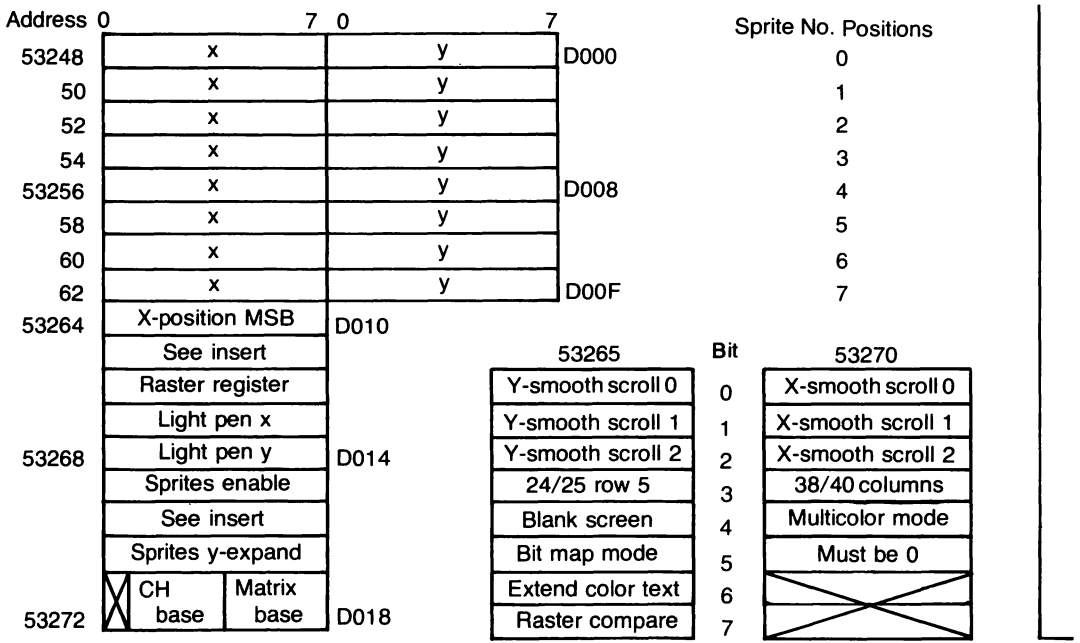
Note: visible window is lines 51 to 251

Set Raster Interrupt Position

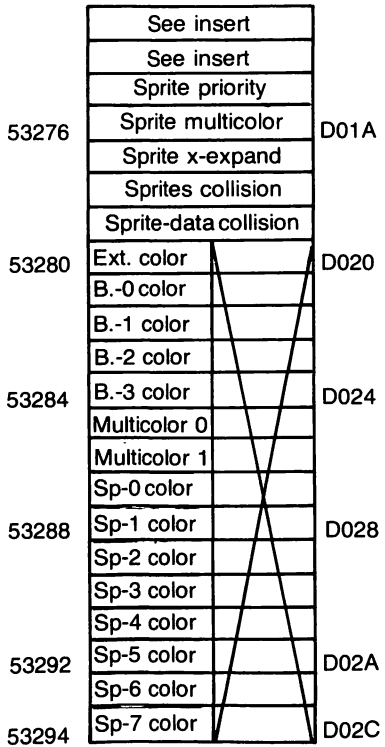
POKE 53266, line (for lines 0-255)

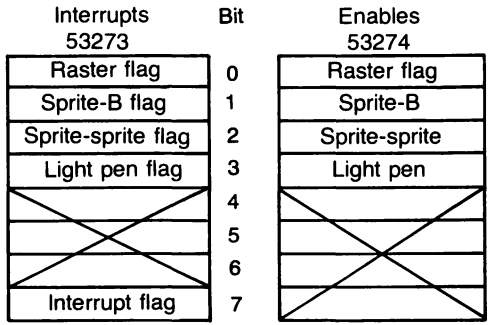
POKE 53265, 128 (if line>255)

VIC II MEMORY MAP



(continued)


See insert
See insert
Sprite priority
53276
Sprite multicolor
D01A
Sprite x-expand
Sprites collision
Sprite-data collision
53280
Ext. color
D020
B.-0 color
B.-1 color
B.-2 color
53284
B.-3 color
D024
Multicolor 0
Multicolor 1
Sp-0 color
53288
Sp-1 color
D028
Sp-2 color
Sp-3 color
Sp-4 color
53292
Sp-5 color
D02A
Sp-6 color
53294
Sp-7 color
D02C


Interrupts
53273
Bit
Enables
53274
Raster flag
0
Raster flag
Sprite-B flag
1
Sprite-B
Sprite-sprite flag
2
Sprite-sprite
Light pen flag
3
Light pen
4
5
6
Interrupt flag
7


Unused

Appendix F

Sound

SOUND GENERATION

Voice Selection. Add V to reference memory location; V=0 for Voice 1; V=+7 for Voice 2; V=+14 for Voice 3

Master Volume. All voices and modes: L=0 to 15, 15=maximum (linear scale) Bits 0-3

```
POKE 54296,(PEEK(54296) AND 240) OR 1
```

Frequency Set. See table for note values; Bits 0-7

```
POKE 54273+V,HI
  HI=INT(frequency/.06097/256)
POKE 54272+V,LO
  LO=frequency/.06097-256*HI
```

Voice Gate. Bit 1

On	POKE 54276+V, PEEK(54276) OR 1
Off	POKE 54276+V, Peek(54276) AND 254

Voice Sustain Level. L=0-15, 15 maximum (linear) Bits 4-7

```
POKE 54278+V,(PEEK(54278+V)AND 15)
OR (16*L)
```

Voice Attack Rate. A=0-15, 0=fastest (see table) Bits 4-7

```
POKE 54277+V, (PEEK(54277+V) AND 15)
OR 16*A
```

Voice Decay Rate. D=0-15, 0=fastest (see table) Bits 0-3

```
POKE 54277+V, (PEEK(54277+V) AND 240)
OR D
```

Voice Release Rate. R=0-15, 0=fastest (see table) Bits 0-3

```
POKE 54278+V, (PEEK(54278+V) AND 240)
OR R
```

Waveform. Bit=4 for Triangle; Bit=5 for Sawtooth; Bit=6 for Pulse; Bit=7 for Noise; Bit number=B

```
On    POKE 54276+V, PEEK(54276+V)
      OR (2↑B)
Off   POKE 54276+V, PEEK(54276+V)
      AND (255-2↑B)
```

Pulse Width. W=INT(% on-time/40.95), W=2048 =square wave

POKE 54275,HI HI=INT(W/256) Bits 0-3
POKE 54274,LO LO=W-256*HI Bits 0-7

Filter Type. Bit=4 for low-pass, Bit=5 for band-pass, Bit=6 for high-pass; Bit number=F

On POKE 54296, PEEK(54296) OR (2↑F)
Off POKE 54296, PEEK(54296) AND (255-2↑F)

Filter Cutoff. C=frequency/5.86

POKE 54294,HI HI=INT(C/16) Bits 0-7
POKE 54293,LO LO=C-16*HI Bits 0-2

Filter Peaking. P=0-15; 0=no peaking; Bits 4-7

POKE 54295, (PEEK(54295) AND 15) OR 16*P

Filter Input Select. I=0=Voice 1, I=1=Voice 2, 1=2=Voice 3, 1=3=External; Bits 0-3

On POKE 54295, PEEK(54295) OR (2↑I)
Off POKE 54295, PEEK(54295) AND (255-2↑I)

Voice 3 Direct. Bit 7

On POKE 54296, PEEK(54296) AND 127) (normal)
Off POKE 54296, PEEK(54296) OR 128

Synchronize. S=0 for Voice 1 to Voice 3, S=7 for

Voice 2 to Voice 1, S=14 for Voice 3 to Voice 2; Bit=1

On	POKE 54276+S, PEEK(54276+S) OR 2
Off	POKE 54276+S, PEEK(54276+S) AND 253

Modulate. M=1 for Voice 1 by Voice 3, M=7 for Voice 2 by Voice 1, M=14 for Voice 3 by Voice 2; Bit=2

On	POKE 54276+M,PEEK(54276+M) OR 4
Off	POKE 54276+M, PEEK(54276+M) AND 251

Test Disable. Bit=3

Disable	POKE 54276+V, PEEK(54276+V) OR 8
Enable	POKE 54276+V, PEEK(54276+V) AND 247

Read Voice 3 Frequency. (Use for random numbers, vibrato, warble, etc.) Bits 0-7

```
PEEK 54299
```

Read Voice 3 Amplitude. (Use for wah-wah, rotating speaker, etc.) Bits 0-7

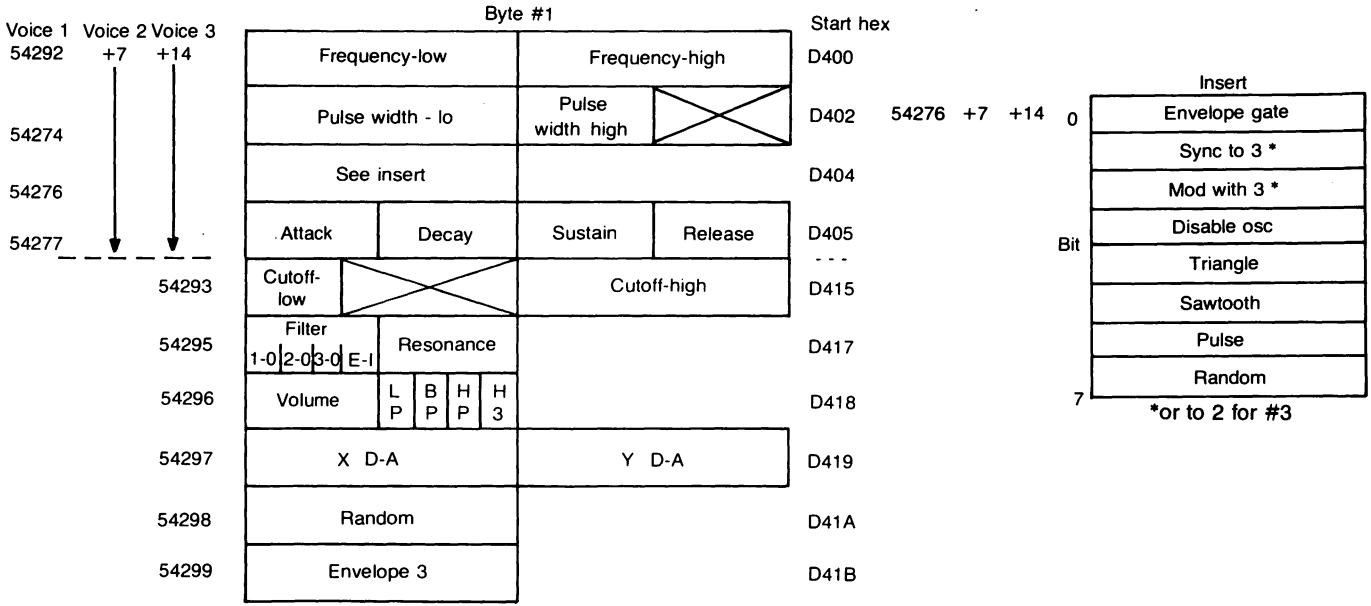
```
PEEK 54300
```

ATTACK, DECAY, AND RELEASE DURATIONS

Poke Value	Attack, A	Decay/Release, D/R
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 sec
11	800 ms	2.4 sec
12	1 sec	3 sec
13	3 sec	9 sec
14	5 sec	15 sec
15	8 sec	24 sec

Note: durations are from zero to maximum level and are in seconds or milliseconds.

SID MEMORY MAP



Value to be poked in decimal

Frequency in hertz = poke-value/0.06097

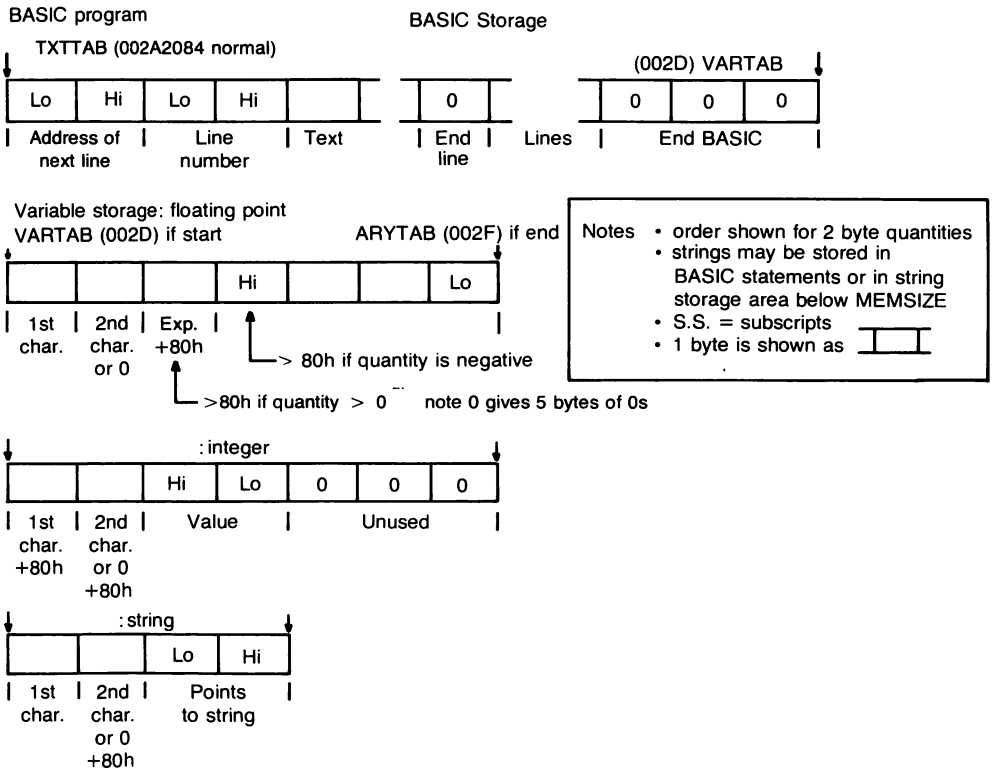
Note	Octave 0	Octave 1	Octave 2	Octave 3	Octave 4	Octave 5	Octave 6	Octave 7
C	268	536	1072	2145	4291	8583	17167	34334
C#	284	568	1136	2273	4547	9094	18188	36376
D	301	602	1204	2408	4817	9634	19269	38539
D#	318	637	1275	2551	5103	10207	20415	40830
E	337	675	1351	2703	5407	10814	21629	43258
F	358	716	1432	2864	5728	11457	22915	45830
F#	379	758	1517	3034	6069	12139	24278	48556
G	401	803	1607	3215	6430	12860	25721	51443
G#	425	851	1703	3406	6812	13625	27251	54502
A	451	902	1804	3608	7217	14435	28871	57743
A#	477	955	1911	3823	7647	15294	30588	61176
B	506	1012	2025	4050	8101	16203	32407	64814

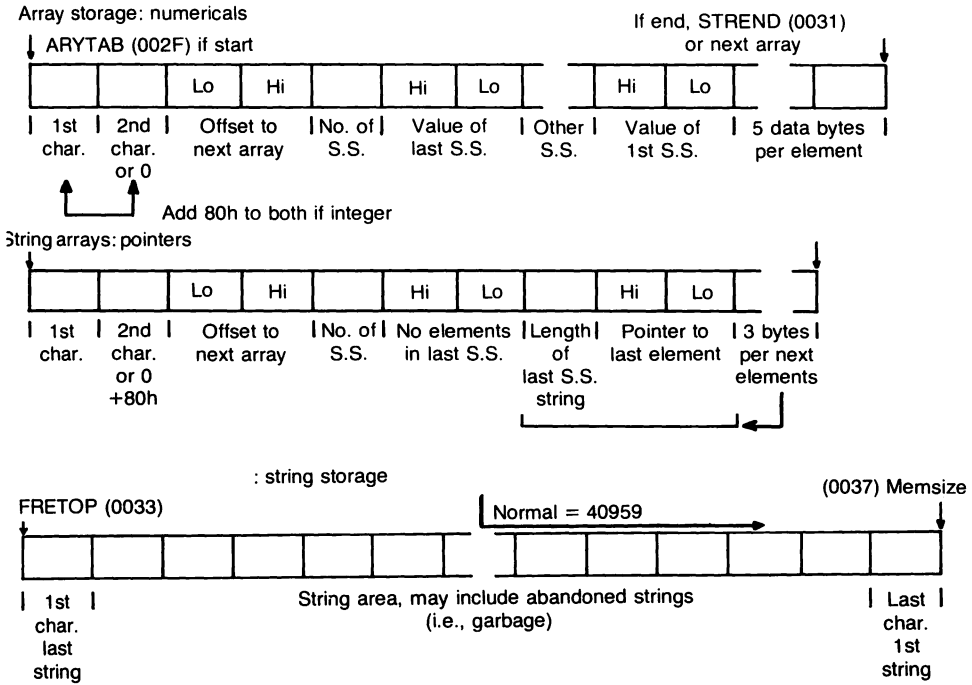
Note \ Octave	0	1	2	3	4	5	6	7
C	1 - 12	2 - 24	4 - 48	8 - 97	16 - 195	33 - 135	67 - 15	134 - 30
C#	1 - 28	2 - 56	4 - 112	8 - 225	17 - 195	35 - 134	71 - 12	142 - 24
D	1 - 45	2 - 90	4 - 180	9 - 104	18 - 209	37 - 162	75 - 69	150 - 139
D#	1 - 62	2 - 125	4 - 251	9 - 247	19 - 239	39 - 223	79 - 191	159 - 126
E	1 - 81	2 - 163	5 - 71	10 - 143	21 - 31	42 - 62	84 - 125	168 - 250
F	1 - 102	2 - 204	5 - 152	11 - 48	22 - 96	44 - 193	89 - 131	179 - 6
F#	1 - 123	2 - 246	5 - 237	11 - 218	23 - 181	47 - 107	94 - 214	189 - 172
G	1 - 145	3 - 35	6 - 71	12 - 143	25 - 30	50 - 60	100 - 121	200 - 243
G#	1 - 169	3 - 83	6 - 167	13 - 78	26 - 156	53 - 57	106 - 115	212 - 230
A	1 - 195	3 - 134	7 - 12	14 - 24	28 - 49	56 - 99	112 - 199	225 - 143
A#	1 - 221	3 - 187	7 - 119	14 - 239	29 - 223	59 - 190	119 - 124	238 - 248
B	1 - 250	3 - 244	7 - 233	15 - 210	31 - 165	63 - 75	126 - 151	253 - 46

Poke-values: High byte—low byte

Middle C

Appendix G
BASIC Program, BASIC Storage


BASIC program
BASIC Storage
TXTTAB (002A2084 normal)
(002D) VARTAB
Lo
Hi
Lo
Hi
0
0
0
0
Address of next line
Line number
Text
End line
Lines
End BASIC
Variable storage: floating point
VARTAB (002D) if start
ARYTAB (002F) if end
Hi
Lo
1st char.
2nd char. or 0
Exp. +80h
> 80h if quantity is negative
>80h if quantity > 0
note 0 gives 5 bytes of 0s
Notes
• order shown for 2 byte quantities
• strings may be stored in BASIC statements or in string storage area below MEMSIZE
• S.S. = subscripts
• 1 byte is shown as
: integer
Hi
Lo
0
0
0
1st char. +80h
2nd char. or 0 +80h
Value
Unused
: string
Lo
Hi
1st char.
2nd char. or 0 +80h
Points to string


Array storage: numericals
ARYTAB (002F) if start
If end, STREND (0031)
or next array
Lo
Hi
Hi
Lo
Hi
Lo
1st char.
2nd char. or 0
Offset to next array
No. of S.S.
Value of last S.S.
Other S.S.
Value of 1st S.S.
5 data bytes per element
Add 80h to both if integer
String arrays: pointers
Lo
Hi
Hi
Lo
Hi
Lo
1st char.
2nd char. or 0 +80h
Offset to next array
No. of S.S.
No elements in last S.S.
Length of last S.S. string
Pointer to last element
3 bytes per next elements
: string storage
FRETOP (0033)
Normal = 40959
(0037) Memsize
1st char. last string
String area, may include abandoned strings (i.e., garbage)
Last char. 1st string

Appendix H

Input and Output

INPUT/OUTPUT INTERRUPTS

Interface number 1, Location 56333
Flag feeds CPU maskable interrupt

Bit no.	Read (is from)	Write (enables)
7	Any interrupt	(Sets flag)
4	Tape read	Serial bus SRQ
3	Serial port	Serial port
2	Clock-alarm	Clock-alarm
1	Timer B	Timer B
0	Timer A	Timer A

Interface number 2, Location 56589
Flag feeds CPU nonmaskable interrupt

7	Any NMI	(Sets flag)
4	RS-232 received	RS-232

3	Serial port	Serial port
1	Timer B	Timer B
0	Timer A	Timer A.

Read Potentiometer. For games, analog to digital conversions, and paddles

Pot X PEEK (54297)
Pot Y Peek (54298)

Range 0-255; 0=0 volts, 255=+5 volt supply (linear scale)

Read Paddles. Not recommended from BASIC; see the *Programmer's Reference Guide*

Read Joystick

A: PEEK(56320) AND 31
B: PEEK(56321) AND 31

Joystick PEEK values

Fire Button Open		Fire Button Pressed
Value	Meaning	Value
0	No Direction	16
1	Up	17
2	Down	18
4	Left	20
8	Right	24

Combinations are possible: 9 = up and right, and so on.

Read number of characters in buffer

```
PEEK(198)
```

Read and write to buffer. B=631-640 (normal)

```
PEEK(B)
POKE B, character
```

Repeat key

all on	POKE 650,128
all off	POKE 650,100
normal	POKE 650,0

Repeat speed. speed= 0-255, normal= 4

```
POKE 651, speed
```

Repeat delay. delay= 0-255, normal=16

```
POKE 652, delay
```

Keyboard buffer size. size=0-10, normal= 10, 0 locks keyboard

```
POKE 649, size
```

Note: buffer is first-in first-out.

Select General Timer Mode. Poke location 56334; add 256 for second timer pair.

Bit no.	Mode	Values
4	Force load	1=Load
3	Run mode	1=One shot, 0=repeat
2	Output to PB6	1=toggle, 0=pulse
1	Output to PB6	1=Yes, 0=no output
0	Start/Stop	1=start, 0=stop

Select Timer A Mode. Poke location 56334; add 256 for second unit.

7	Clock rate	1=50 Hz, 0=60 Hz
6	Serial port use	1=Output, 0=Input
5	Timer counts	1=CNT input, 0=System 1 MHz Clock

Select Timer B Mode. Poke location 56335; add 256 for second unit.

7	Set	1=Alarm, 0=Clock
6,5	Select input	00=System 1 MHz clock 01=Positive CNT Input 10=Timer R underflow (Cascade) 11=Timer R underflow gated by CNT

Clock Set. Add 256 for second clock

POKE 56587	Set hours; add 128 for PM
POKE 56586	Set minutes
POKE 56585	Set seconds
POKE 56584	Set 1/10 second and start clock

Device Select (see OPEN)

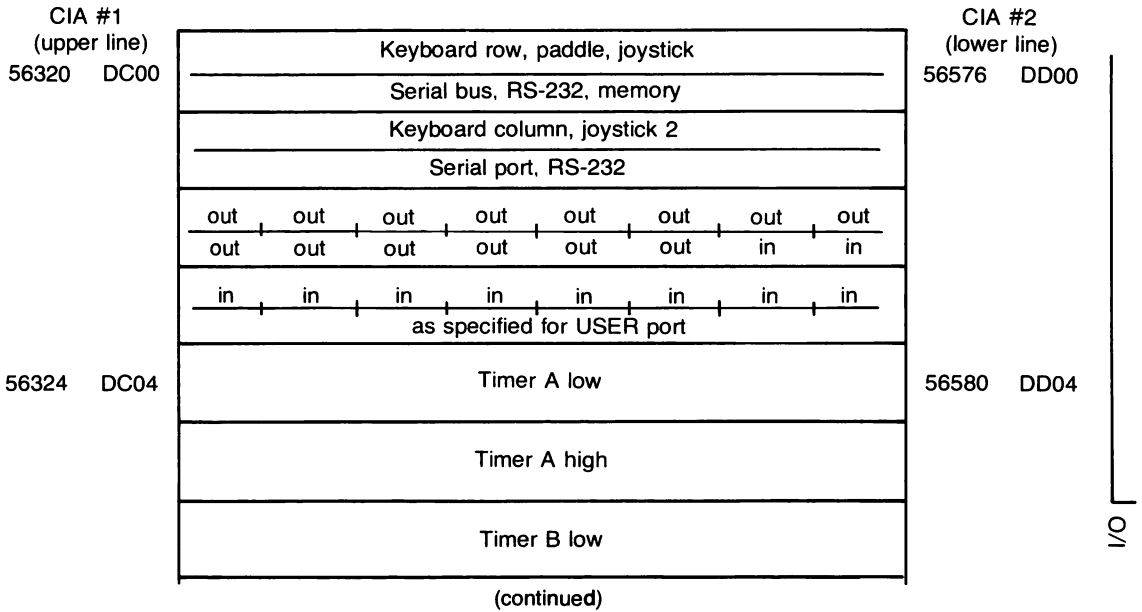
0	Keyboard	(default for input)
1	Tape	
2	Modem	
3	Screen	(default for output)
4,5	Printer	
8 to 11	Disk	

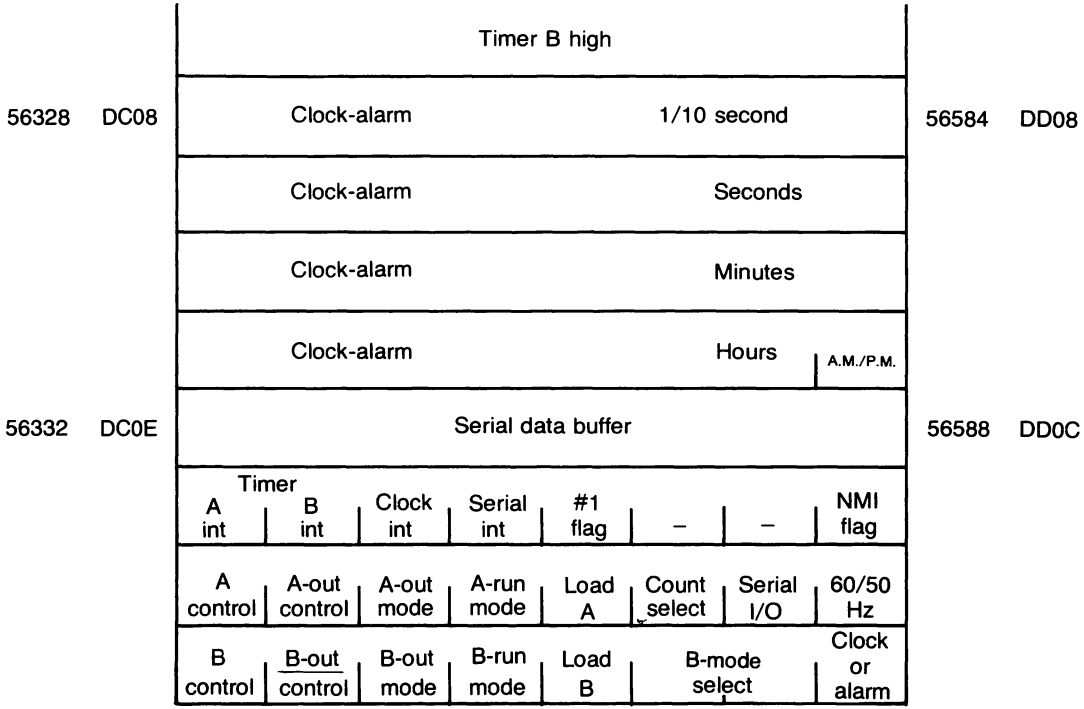
Default Select

Input- POKE 153, device 0=normal
Output-POKE 154, device 3=normal

Device Status; see ST, or PEEK (144)

Bit	Serial I/O	Tape Read	Tape Write
0	Write time-out		
1	Read time-out		
2		Short block	Short block
3		Long block	Long block
4		Major error	Any mismatch
5		Checksum error	Checksum error
6	EOI line	EOF	
7	Device not present	End of tape	End of tape.


CIA #1
(upper line)
56320 DC00
Keyboard row, paddle, joystick
Serial bus, RS-232, memory
Keyboard column, joystick 2
Serial port, RS-232
out out out out out out out out
out out out out out out in in
in in in in in in in in
as specified for USER port
56324 DC04
Timer A low
Timer A high
Timer B low
(continued)
CIA #2
(lower line)
56576 DD00
56580 DD04
I/O


Timer B high
56328 DC08
Clock-alarm
1/10 second
56584 DD08
Clock-alarm
Seconds
Clock-alarm
Minutes
Clock-alarm
Hours
A.M./P.M.
56332 DC0E
Serial data buffer
56588 DD0C
Timer A int
Timer B int
Clock int
Serial int
#1 flag
–
–
NMI flag
A control
A-out control
A-out mode
A-run mode
Load A
Count select
Serial I/O
60/50 Hz
B control
B-out control
B-out mode
B-run mode
Load B
B-mode select
Clock or alarm

Appendix I
Codes

CODE	SET 1	Screen SET 2	ASCII	KEY Token
0	@			1
1	A	a		3
2	B	b		5
3	C	c		7
4	D	d		9
5	E	e	WHT	+
6	F	f		£
7	G	g		DEL
8	H	h	DSC	←
9	I	i	ESC	W
10	J	j		R
11	K	k		Y
12	L	l		I
13	M	m	RET	P
14	N	n	L.C.	*
15	O	o		RET
16	P	p		
17	Q	q	↓CSR	A
18	R	r	RON	D
19	S	s	CLR	G
20	T	t	DEL	J
21	U	u		L
22	V	v		;
23	W	w		CSR

See Programmers Manual for exact forms of diagonal entries

CODE	Screen SET 1	Screen SET 2	ASCII	KEY	Token
24	X	x		STP	
25	Y	y			
26	Z	z		X	
27	[			V	
28	£		RED	N	
29	]		CSR →	,	
30	↑		GRN	/	
31	←		BLU	CSR ↓	
32	SP		SP	SP	
33	!		!	Z	
34	"		"	C	
35	#		#	B	
36	$		$	M	
37	%		%		
38	&		&		
39	'		'	F1	
40	(		(		
41	)		)	S	
42	*		*	F	
43	+		+	H	
44	,		,	K	
45	−		−	:	
46	.		.	=	
47	/		/	F3	
48	0		0	Q	
49	1		1	E	
50	2		2	T	
51	3		3	U	

Screen CODE	SET 1	SET 2	ASCII	KEY	Token
52	4		4	O	
53	5		5	@	
54	6		6	↑	
55	7		7	F5	
56	8		8	2	
57	9		9	4	
58	:		:	6	
59	;		;	8	
60	<		⊂	0	
61	=		=	–	
62	>		⊃	HOM	
63	?		?	F7	
64	⊟		@		
65	♠	A	A		
66	▯	B	B		
67	⊟	C	C		
68	▭	D	D		
69	▭	E	E		
70	▭	F	F		
71	▯	G	G		
72	▯	H	H		
73	◱	I	I		
74	◱	J	J		
75	◱	K	K		
76	▢	L	L		
77	▨	M	M		
78	▨	N	N		

See Programmers Manual for exact forms of diagonal entries

CODE	SET 1	Screen SET 2	ASCII	KEY	Token
79		O	O		
80		P	P		
81		Q	Q		
82		R	R		
83		S	S		
84		T	T		
85		U	U		
86		V	V		
87		W	W		
88		X	X		
89		Y	Y		
90		Z	Z		
91			[		
92			£		
93			]		
94			↑		
95			←		
96	SP				
97					
98					
99					
100					
101					
102					
103					
104					
105					
106					

CODE	Screen SET 1	Screen SET 2	ASCII	KEY	Token
107					
108					
109					
110					
111					
112					
113					
114					
115					
116					
117					
118					
119					
120					
121					
122					
123					
124					
125					
126					
127					
128	@				END
129	A	a			FOR
130	B	b			NEXT
131	C	c			DATA
132	D	d			INPUT #

 Indicates character/background are reversed

CODE	SET 1	Screen SET 2	ASCII	KEY	Token
133	E	e			INPUT
134	F	f			DIM
135	G	g			READ
136	H	h			LET
137	I	i			GOTO
138	J	j			RUN
139	K	k			IF
140	L	l			RESTORE
141	M	m	RET		GOSUB
142	N	n	UC		RETURN
143	O	o			REM
144	P	p	BLK		STOP
145	Q	q			ON
146	R	r	CSR ↑		WAIT
147	S	s	RVS ON		LOAD
148	T	t	CLR		SAVE
149	U	u	DEL		VERIFY
150	V	v			DEF
151	W	w			POKE
152	X	x			PRINT #
153	Y	y			PRINT
154	Z	z			CONT
155	[				LIST
156	£		PUR		CLR
157	]		CSR ←		CMD
158	↑		YEL		SYS
159	←		CYN		OPEN
160	SP	SP			CLOSE

CODE	Screen SET 1	Screen SET 2	ASCII	KEY	Token
161	!				GET
162	"				NEW
163	#				TAB(
164	$				TO
165	%				FN
166	&				SPC(
167	'				THEN
168	(				NOT
169	)				STEP
170	*				+
171	+				–
172	,				*
173	-				/
174	.				1
175	/				AND
176	0				OR
177	1				>
178	2				=
179	3				<
180	4				SGN
181	5				INT
182	6				ABS
183	7				USR
184	8				FRE
185	9				POS
186					SQR
187	:				RND

 Indicates character/background are reversed

CODE	SET 1	Screen SET 2	ASCII	KEY	Token
188					LOG
189					EXP
190					COS
191					SIN
192					TAN
193		A			ATN
194		B			PEEK
195		C			LEN
196		D			STR$
197		E			VAL
198		F			ASC
199		G			CHR$
200		H			LEFT$
201		I			RIGHT$
202		J			MID$
203		K			
204		L			
205		M			
206		N			
207		O			
208		P			
209		Q			
210		R			
211		S			
212		T			
213		U			
214		V			
215		W			

CODE	SET 1	Screen SET 2	ASCII	KEY	Token
216					
217					
218					
219					
220					
221					
222					
223					
224	SP				
225					
226					
227					
228					
229					
230					
231					
232					
233					
234					
235					
236					
237					
238					
239					
240					
241					
242					

 Indicates character/background are reversed

	Screen				
CODE	SET 1	SET 2	ASCII	KEY	Token
243					
244					
245					
246					
247					
248					
249					
250					
251					
252					
253					
254					
255					

Appendix J
Op Codes

		Addressing Mode										
Instruction	Short Description	Immediate	Zero Page	Zero Page, X	Absolute	Absolute, X	Absolute, Y	(Indirect, X)	(Indirect), Y	Accumulator	Implied	Relative
ADC	Add A	69	65	75	6D	7D	79	61				
AND	And A	29	25	35	2D	3D	39	21	31			
ASL	Shift L 1		06	16	0E	1E				0A		
BCC	Branch Carry											90
BCS	Branch Carry											B0
BEQ	Branch 0											F0
BIT	Test Bit		24		2C							
BMI	Branch—											30
BNE	Branch ≠ 0											D0
BPL	Branch +											10
BRK	Break										00	
BVC	Branch O'Flo											50
BVS	Branch O'Flo											70
CLC	Clear Carry										18	
CLD	Clear Decimal										D8	
CLI	Clear I Flag										58	
CLV	Clear 0 Flag										B8	
CMP	Compare A	C9	C5	D5	CD	DD	D9	C1	D1			
CPX	Compare X	E0	E4		EC							
CPY	Compare Y	C0	C4		CC							
DEC	Decrement		C6	D6	CE	DE						
DEX	Decrement X										CA	
DEY	Decrement Y										88	
EOR	XOR A	49	45	55	4D	5D	59	41	51			
INC	Increment		E6	F6	EE	FE						
INX	Increment X										E8	
INY	Increment Y										C8	
JMP	Jump				4C (6C)							
JSR	Jump w. Return				20							
LDA	Load A	A9	A5	B5	AD	BD	B9	A1	B1			
LDX	Load X	A2	A6	*B6	AE		BE					
LDY	Load Y	A0	A4	B4	AC	BC						
LSR	Shift Right		46	56	4E	5E				4A		
NOP	No Op										EA	
ORA	OR A	09	05	15	0D	1D	19	01	11			
PHA	Push A										48	
PHP	Push Sta.										08	
PLA	Pull A										68	

Addressing Mode

Instruction	Short Description	Immediate	Zero Page	Zero Page, X	Absolute	Absolute, X	Absolute, Y	(Indirect, X)	(Indirect), Y	Accumulator	Implied	Relative
PLP	Pull Sta.										28	
ROL	Rotate Left		26	36	2E	3E				2A		
ROR	Rotate Right		66	76	6E	7E				6A		
RTI	Ret. Int.										40	
RTS	Ret. S.R.										60	
SBC	SUB. A	E9	E5	F5	ED	FD	F9	E1	F1			
SEC	Set Carry F.										38	
SED	Set Decimal										F8	
SEI	Sel Int. F.										78	
STA	STO A		85	95	8D	9D	99	81	91			
STX	STO X		86	*96	8E							
STY	STO Y		84	94	8C							
TAX	A → X										AA	
TAY	A → Y										A8	
TSX	SP → X										BA	
TXA	X → A										8A	
TXS	X → SP										9A	
TYA	Y → A										98	

Number of Bytes	2	2	2	3	3	3	2	2	1	1	2

( ) indirect
* Zero Page, Y

Computer Companion for the Commodore 64™

by Robert P. Haviland

- Instant access to all the information you need to create programs and get them running smoothly!
- All the key words used by the Commodore 64, listed in easy-to-use alphabetical order!
- Includes all functions, commands, and statements with class designation, required form, conditions and results of use, and token used for internal storage!
- Each key word is listed on a separate page . . . with tab index for quick reference!
- Lie-flat comb binding for easy use . . . printed in large, easy-to-read type on heavy card stock!

Now, you can have all the information you need for programming your C-64 right at your fingertips . . . when and where you need it. This exceptional sourcebook is designed for quick and easy reference . . . conveniently arranged to refresh your memory on available terms and the specifics of how those terms should be written.

Never again will you have to shuffle through a pile of BASIC manuals searching for a particular term to use in a program. What you need is right here, ready to use almost instantly.

Included are listings of reserved variables, operators, codes, important memory locations, colors, tones, and the assembly language instruction set for the 6402 microprocessor.

If you own or have access to a Commodore 64—this is the most practical computing helpmate you can have!

Robert P. Haviland is a professional engineer with extensive experience in the design, construction, and operation of home computers. He has written several bestselling computer books for TAB.

TAB BOOKS Inc.
Blue Ridge Summit, Pa. 17214

FPT > 11.95 ISBN 0-8306-1913-5

PRICES HIGHER IN CANADA 1145-1284