

1613

# COMPUTER COMPANION FOR THE VIC-20<sup>®</sup>



BY ROBERT P. HAVILAND



**COMPUTER COMPANION**  
**FOR THE**  
**VIC-20<sup>®</sup>**

**BY ROBERT P. HAVILAND**

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

FIRST EDITION

FIRST PRINTING

Copyright © 1983 by TAB BOOKS Inc.

Printed in the United States of America

Reproduction or publication of the content in any manner, without express permission of the publisher, is prohibited. No liability is assumed with respect to the use of the information herein.

Library of Congress Cataloging in Publication Data

Haviland, Robert P.  
Computer companion for the VIC-20.

Includes index.  
1. VIC 20 (Computer) I. Title.  
QA76.8.V5H38 1983 001.64 83-4963  
ISBN 0-8306-0613-0 (pbk.)

# Contents

---

<b>Introduction</b>	<b>v</b>
<b>Reserved Words, organized alphabetically</b>	<b>1</b>
<b>Appendix A Reserved Variables</b>	<b>85</b>
<b>Appendix B Operators</b>	<b>87</b>
<b>Appendix C VIC-20 Codes</b>	<b>89</b>
<b>Appendix D VIC-20 Memory Map</b>	<b>101</b>
<b>Appendix E VIC-20 Important Memory Locations</b>	<b>103</b>
<b>Appendix F VIC-20 Screen Memory Addresses</b>	<b>105</b>
<b>Appendix G VIC-20 COLORS</b>	<b>107</b>
<b>Appendix H VIC-20 Tone Values</b>	<b>109</b>
<b>Appendix I Hex-Decimal Converter</b>	<b>111</b>

**Appendix J**    **Relative Jump-Count Converter**    **113**

**Decimal-Hex**

**Appendix K**    **6501-6505 Operation Codes**    **115**



## Introduction

---

This little manual is intended to be a constant companion to the VIC-20 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 key word. All key words used by CBM BASIC, version 2.0 are included, arranged alphabetically. For each key word you are given:

- The *Name*, if different from the key word
- The *Token* used for internal storage
- The *Class* of instruction
- The required *Form*
- The *Conditions* and results of use

In addition, examples, error message interpretations, warnings, and cautions are given for some key words.

The following terminology has been used:

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

Example: SIN X.

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

Example: STOP.

*Statements*—a statement provides information to the computer, usually relating to program execution. Statements are sometimes grouped as declarative, conditional, and imperative, which include commands. Some statements are labeled as input-output (I/O) statements. The division followed is that used in the Commodore manuals.

Example: DATA.

*Expression*—an expression is a combination of statements, variables, functions, constants, and/or operators that are reduced to a single value 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 run-time follows line 10 run-time.

Functions and expressions are understood to be numerical unless specified. Other classes are logical, string, and graphic. Numericals can be *integer*, having no decimal point and no fraction, or *floating point*, allowing a decimal point and fractional parts. Floating point numbers may include exponents, designated by E. The symbol % indicates integers, and the symbol \$ indicates strings.

There are appendices giving summary information on delimiters, operators, and priorities as well as other data useful in programming, including machine-language codes and forms.

In preparing this material, the ultimate authority has been the VIC-20 computer itself, in 5K form. The manufacturer's literature is the next level of authority, and the general literature the last. All stated forms have been checked and are correct at the time of writing.

Remember that some items are computer specific, for example, the memory map. The key words are valid for any computer using CBM Basic, version 2.0. The 6502 machine codes are valid for several other popular computers.

Note should be made of two conventions. A letter enclosed in a circle indicates that the shift key and that letter key are pressed together. In the symbol table, a symbol with a diagonal through it indicates that the symbol prints as white on a black background, that is, is reversed. Other conventions follow standard Commodore practice.

The author, editor, and publisher would appreciate a note if you find any errors that may have crept in, indications of changes by the manufacturer, or suggestions for making the volume more useful.

**Token** 182

**Name** Absolute Value

**ABS**

**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, with two exceptions, the first parenthesis is not part of the key word.

**Token** 175

**Name** Logical AND

**AND**

**Class** Logical Operator

**Alternate** A(N)

# AND

**Form** Value 1 AND Value 2

**Conditions** Returns the logical AND of the corresponding bits of the values according to the following rules:

$$0 \text{ AND } 1 = 0$$

$$1 \text{ AND } 0 = 0$$

$$0 \text{ AND } 1 = 0$$

$$1 \text{ AND } 1 = 1$$

Values are truncated to 16 bit 2's complement integers before the AND is done.

Values must be numeric.

**Note** Commonly used in statements of the form:

IF A = 10 AND B = 0 THEN . . . . .

**Token** 198

**Name** ASCII CODE

**Class** Function

**Alternate** A(S)

**ASC**

# 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** Arc tangent

**Class** Function

**Alternate** A<sup>(T)</sup>

ATN

## ATN

**Form** (value)

**Conditions** Returns the angle whose tangent is the value i.e., the arc tangent 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 type (the evaluation is floating point).

ATN has no meaning for string quantities.

**Example** ATN(1) = .785398163.

**Token** 199                      **Name** Character Code

**Class** String Function              **Alternate** C(H)\$

## CHR\$

CHR\$

**Form** CHR\$(value)

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

The value must be in the range 0 to 255 (some codes do not print, and some are duplicated—see appendix).

The value may be integer or floating point, but floating point values are rounded down to the nearest integer.

CHR\$ has no meaning for string quantities.

**Example** CHR\$(99.999) = CHR\$(99) = R

**Note** Keys F1-F8, represented by CHR\$ 133 – CHR\$ 140, may be assigned a numerical or string value. This may be done in a key-scan routine or by variable assignment.

**Caution** The \$ is part of the short form.

**Token** 160

**Name** Close File

**Class** I/O Statement

**Alternate** CL⓪

# CLOSE

CLOSE

**Form** CLOSE file number

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

CLOSE discards any data in the associated file buffers; to prevent loss of data, use PRINT# prior to CLOSE.

See OPEN

**Example** CLOSE 4

Closes the channel to the printer identified as device number four.

**Note** No error code is given if the device terminated is not present.



**Token** 156

**Name** Clear variables

**Class** Statement

**Alternate** C(L)

## CLR

**Form** CLR

**Conditions** Clears out all variables, undimensions arrays, and sets the data pointer to start.

CLR

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

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

**Hint** For long programs, initialize needed variables using direct mode input; then initiate program execution by using GOTO 0. This saves program space.

If variables are to be preserved for a rerun, initiate the next execution with a GOTO 0.

**Token** 157      **Name** Change Output Mode

**Class** I/O Statement      **Alternate** C(M)

## CMD

**Form** CMD file number

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

Use to send a listing or a block of data to the device (tape, disk, printer, etc.)

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

See OPEN, CLOSE

**Error Codes** ?FILE NOT OPEN

?NOT OUTPUT FILE

CMD

**Token** 154

**Name** Continue Execution

**Class** Command

**Alternate** CⓄ

# CONT

**Form** CONT

**Conditions** If program has been suspended by STOP, the stop key or by END within the program, CONT causes execution to resume at the next statement.

Use STOP to insert breaks in programs for debugging, examining data, and so on, and CONT to resume.

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

**Error Code** CAN'T CONTINUE

CONT

**Token** 190

**Name** Cosine

**Class** Function

**Alternate** -none-

# COS

**Form** COS (angle)

**Conditions** Returns the cosine of the angle which is in radians.

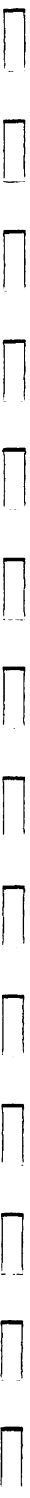
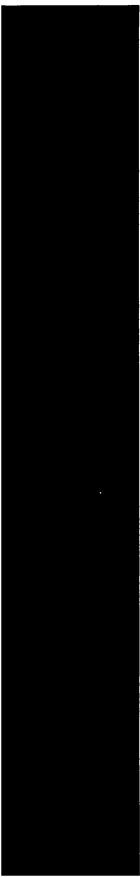
The angle may be a number, a variable, or an expression.

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

COS has no meaning for string quantities.

**Example** COS(1) = .540302306

COS



**Token 131**

**Class Statement**

**Alternate**

D(A)

# DATA

**Form** DATA value 1, value 2, - - -

**Conditions** Holds data for a READ statement.

May hold a single datum (value) or any number of values.

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

Value must correspond in type to the type expected by READ.

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

Spaces, colons, or commas, used as values must be in quotes.

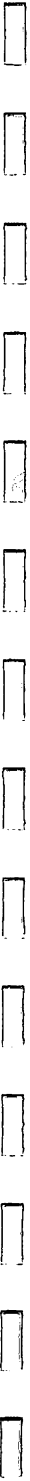
A set of values may be placed in two or more DATA statements. The order must be maintained.

**Hint** Programs execute faster if DATA

**DATA**

statements are placed at the end of the program.

**Error Code** Out of Data



**Token** 150                      **Name** Define Function

**Class** Statement                      **Alternate** DⒺ

## 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 variable name composed of a letter followed by 0 or 1 character, or it can be omitted.

If the name is used, the variable is a dummy replaced by the actual variable at call.

If the name is not used, the variable becomes the name.

The formula does not have to include variables.

DEF FN has no meaning for string quantities.

**Caution** DEF FN must be executed before it is called; the easy way to do this is to place all such definitions at the start of the program.

DEF

**Example** DEF FN AB(X) = EXP(X) + 1

See FN



**Token** 134

**Name** Dimension Array

**Class** Statement

**Alternate** D①

## DIM

**Form** DIM name (elements 1, elements 2 ---)

**Conditions** Reserves memory space for the array of the name given (any legal variable name).

The number of dimensions is equal to the number of element values in parentheses.

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

There is no limit on the number of dimensions or elements, up to the limit of available memory.

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

To be effective, DIM must be executed before the name array is used.

Arrays cannot be redimensioned during program execution.

DIM

If a named array that has not been DIMensioned is encountered during execution, it is assigned 11 elements in each dimension; if a DIM statement is encountered after this, an error results.

More than one array may be dimensioned in a single statement by using the following form:

DIM name (elements 1, elements ---),  
name (elements 1, elements 2 . . . )  
name (elements 1, elements 2) etc.

**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)

Each item in the array can be assigned a unique value.

**Error Code** OUT OF MEMORY

RE DIM'D ARRAY

**Token** 128

**Class** Statement

**Alternate**

E **N**

# END

**Form** END

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

Does not affect the variables or the program.

CONT will initiate execution of any remaining program lines.

END is not required; execution termination and return to keyboard control also occurs when the last program line is executed.

**Note** STOP, and END or last line completion differ only in the report code. The normal use of END is at the division between the main program and subprograms. However, a program with many subroutine calls will execute faster with the subroutines at the start of the program.

**END**

**Token** 189

**Name** e-Exponent

**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 88.02969191.

EXP



**Token** 165

**Name** Function

**Class** Statement

**Alternate** None

## FN

**Forms** DEF FN name (variable) = formula

FN name (variable)

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

(b) Used to call the defined function when needed. When a function definition has been executed, the function is available in program or direct modes.

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

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

See DEF.

**Example** If DEF FN AB(X) = EXP(X) + 1; PRINT FN AB(2) gives 8.38905611.

FN

**Token** 129

**Class** Statement

**Alternate** F⊙

# FOR

**Form** FOR variable-name = initial value TO limit STEP increment

**Conditions** The first element of a FOR-NEXT loop.

The variable name can be any simple numeric variable (not an array).

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 is increased by the increment, or by +1 if no increment is given; the new value is compared to the limit; if it is less, execution returns to the statement following the FOR definition; if it is more, execution proceeds with the statement following NEXT.

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

A NEXT is required for each FOR, or a program error results; the NEXTs can

FOR

be on one line, separated by colons; the form NEXT variable name 1, variable name 2 can be used.

The initial, limit, and increment values may be numbers, variables, or expressions that have integer or floating point values.

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

A FOR-NEXT loop can be run in immediate mode if all elements can be included in an 88 character line.

See NEXT, TO; STEP

**Example** For N = 1 TO 10: PRINT "X": NEXT  
prints ten Xs in a column.

**Token** 184

**Name** Free Memory

**Class** Function

**Alternate** F®

## FRE

**Form** FRE (any value)

**Conditions** Returns the number of 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.

**Caution** The number returned is the number of bytes free at the execution of the FRE. A program can require more memory during its execution, due to encountered variables or DIM statements, and the need for stacks.

FRE





**Token** 161

**Class** Statement

**Alternate** GⒺ

## GET#

**Form** GET# file number, variable  
GET variable

**Conditions** a) Receives one byte of data from the device having the opened file number and assigns it to the variable.

b) If the # and file number are omitted, GET receives one byte from the keyboard and assigns it to the variable; however, it does not wait for key depression.

If no data is available at execution, GET returns the null value.

**Note** In contrast to INPUT#, GET receives only one character, but it will accept special characters such as a carriage return or a quotation mark.

GET#



**Token** 203

**Class** Statement

**Alternate** None

# GO

**Form** GO TO

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

See GOTO.

**GO**



**Token** 141

**Name** Go to Subroutine

**GOSUB**

**Class** Statement

**Alternate** GOⓈ

# GOSUB

**Form** GOSUB line number

**Conditions** Transfers program control to the line number, to execute a subroutine which 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, execution is resumed at the next statement following the GOSUB statement.

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

**Error Message** UNDEF'D STATEMENT if the designated line number is not present or is nonexistent.

**Token** 137

**GOTO**

**Class** Statement

**Alternate** G⊙

# GOTO

**Forms** 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 Message** UNDEF'D STATEMENT if the designated line number is not present or is nonexistent.

**Token** 139

**Class** Statement

**Alternate** None

**IF**

## IF

**Forms** IF condition THEN action

IF condition THEN line number

**Conditions** IF sets up a branching test, the IF-THEN construct.

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

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

The action may be any executable BASIC command or statement. For the special case of a jump to a line number, only the line number is required, the GOTO being optional. The line number must exist.

If the condition evaluates as false, i.e., 0, all remaining elements on the *line* are ignored, the 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, since they may never be implemented.

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

**Token** 133      **Name** Input From Keyboard

**Class** Statement      **Alternate** None

# INPUT

INPUT

**Form** INPUT "prompt"; variable list

**Conditions** Used to secure information from the keyboard.

The prompt is optional; if used, it must conform to string rules. It will be printed on the screen and followed by a question mark and a blinking cursor. If the prompt is omitted, only the question mark and the blinking cursor appear.

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

The response to the screen prompt or question mark should be the values for the variables specified by the variable list, correct in order, number, and type, and separated by commas; a missing quantity will cause a double question-mark to be displayed; wrong types and extra quantities will cause an error message; however, the excess quan-

titles will be ignored; the return key must be pressed after the input is typed.

A null input is interpreted as 0 if a number is expected.

**Cautions** The only way to escape from INPUT is by holding down the run-stop key and pressing the restore key. Variables are not cleared by this.

A skipped quantity will produce an input error, as BASIC assumes that only the last item on the variable list is missing.

**Error Messages** REDO FROM START is probably a type mismatch.

?? indicates a missing quantity.

EXTRA IGNORED indicates that the input includes more quantities than expected.



**Token**                      **Name**    Input From Device

**Class**    I/O Statement                      **Alternate**    I(N)#

## INPUT#

INPUT#

**Form**    INPUT# file number, variable list

**Conditions**    Accepts input from the device specified by a previous OPEN I/O statement.

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

May be used to accept data from the keyboard without displaying a prompt.

May be used to read any whole line of text from the screen; the last screen character will be read as CHR\$(13), or carriage return.

See INPUT, OPEN.

**Use**    Input# is usually the fastest and easiest way to retrieve file data from tape or disk.

**Cautions**    A type error will halt execution

A CHR\$(13), comma, colon, or semi-colon not included in quotes will be read as the end of the variable.

The # is part of the short form.



**Token** 181                      **Name** Integer Function

**Class** Function                      **Alternate** None

## INT

**Form** INT (quantity)

INT

**Conditions** Returns the integer value of the quantity, i.e., the largest integer that is equal to or less than the quantity.

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

INT has no meaning for string quantities.

**Token** 200

**Name** Left String Slice

**Class** String Function

**Alternate** LE(F)\$

## LEFT\$

### LEFT\$

**Form** LEFT\$ (string, length)

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

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

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

See RIGHT\$, MID\$

If length is greater than the number of characters in the string, the entire string is returned.

If length is zero, the null string is returned.

LEFT\$ has no meaning for arithmetic quantities, (but numbers are allowable string elements).

**Caution** The \$ is part of the short form.

**Token** 195 **Name** String Length

**Class** Function **Alternate** None

## LEN

**Form** LEN (String)

**Conditions** Returns the length of the specified string, indicating the number of characters, including nonprinting characters and blanks.

LEN

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

LEN has no meaning for numerical quantities, but numbers are allowable string elements.

**Token** 136

**Class** Statement

**Alternate** L (E)

## LET

**Forms** LET variable = value  
Variable = value

**Conditions** The assignment statement, used to set a variable to a value.

Variable may be any legal variable name, numerical or string.

Value must match 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 form LET A = B = C is not allowed.

LET

**Token** 155

**Class** Command

**Alternate** L①

## LIST

**Forms** LIST

LIST line number 1—line number 2

**Conditions** Used to place a listing of the program in memory on the screen.

LIST

If a single line number is given, only that line number is listed.

If two line numbers separated by a dash are given, the listing starts with the first and proceeds to the last; if a line number does not exist, the next larger line at start and next smaller one at end is used.

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.

The list is edited as it is listed: tokens are expanded and a blank is inserted



after each line number. Other blanks are either removed or inserted.

To slow the screen scroll during list, hold down CTRL. To stop, use RUN-STOP.

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

See CMD for LIST printing.

**Token** 147

**Class** Command

**Alternate** L⊙

# LOAD

**Forms** LOAD

LOAD "filename", device, command.

**Conditions** When used alone, LOAD initiates transfer of the first program encountered from the tape recorder to memory; any program in memory is lost; status messages are given.

**LOAD**

To load a particular program from tape or any program from other devices a filename and device must be specified.

The filename may be a string in quotes, a string variable, or a star in quotes if the first program is wanted; a star is an allowable rightmost part of a truncated name and means that any characters which follow should be ignored. The device may be specified by number or by a numeric variable; if no number is given, tape is assumed.

The command 1 means to load in the original place in memory; if omitted, the

program is placed at the start of the program area.

A LOAD construct with a line number (not a 1) can be used as a command to find a program, LOAD it and RUN it.

Variables from a preceding program are not erased at LOAD and are available if the loaded program is shorter than or the same length as the old, if it is longer, the variables are overwritten.

Use NEW before LOAD if variables must be cleared, or follow LOAD by CLR.

**Examples** LOAD  
LOAD "HELLO"  
LOAD "\*"  
LOAD "HELLO", 1, 1

**Token** 188                      **Name** Natural Logarithm

**Class** Function                      **Alternate** None

## LOG

**Form** LOG (value)

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

Value may be given as a number, a variable, or an expression, and must be greater than zero.

LOG has no meaning for string quantities.

LOG

**Example** LOG (2.5) = .916290732

**Token** 202

**Name** Middle String Slice

**Class** String Function

**Alternate** M①\$

## MID\$

**Form** MID\$ (string, start, length)

**Conditions** Returns the substring of a string from the character number indicated by start and of the number of characters given by length.

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

Start and length may be a number, a variable, or an expression, and must be in the range 0-255.

If length is omitted, or there are fewer characters than specified length to the right of start, 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, but numbers are allowable string elements.

MID\$

**Example** MID\$ ("QWER" + "ASDF", 2, 4) =  
ERAS

**Caution** The \$ is part of the short form.



**Token** 162

**Class** Command

**Alternate** None

## **NEW**

**Conditions** Erases the program, variables, and stacks currently in memory in preparation for a new program.

If encountered in a program, terminates execution and erases program, variables and stacks.

See LOAD.

**NEW**



**Token** 130

**Class** Statement

**Alternate** NⒺ

## NEXT

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

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

If a variable name is given, NEXT terminates the loop for that variable.

Several variable loops may be terminated by including their names, separated by commas, after NEXT. If this form is not used, there must be a NEXT for each FOR.

If no variable is specified, NEXT is assumed to apply to the most recent FOR.

If there is no NEXT for a given FOR, a search to the end of the program is made, and execution is halted.

**Error Messages** READY may indicate a missing NEXT.

NEXT without FOR indicates an excess NEXT.

NEXT



**Token** 168

**Name** Logical NOT

**Class** Logical Operator

**Alternate** NⓈ

# NOT

**Form** NOT value

**Conditions** Returns the logical NOT of the bits of the value according to the following rules:

NOT 1 = 0

NOT 0 = 1

Value is truncated to 16 bit 2's complement integer before the NOT is done.

Value must be numeric.

**NOT**



**Token** 159      **Name** Open Device Channel

**Class** I/O Statement      **Alternate**   P

# OPEN

**Form** OPEN file number, device number, command number, string.

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

The file number may be any number between 1 and 255, and is required.

The device number selects as follows;

- None given - Tape deck
- 0 - Keyboard
- 1 - Tape deck
- 2 - RS-232 device
- 3 - Screen
- 4,5 - Printer
- 8 - Disk
- 4-127 - Serial bus device, (CR)
- 128-255 - Serial bus device, (CR/LF)

The command number is specific to each device. Common ones are:

Tape    0      Read file

**OPEN**

	1	Write file
	2	Write file, plus EOT
Disk	1-14	Open data channel
	15	Open command channel
Printer	0	Uppercase/graphics
	7	Upper/lowercase

The string, in quotes, is used as the filename for tape, and may be used as the filename or as a command for Disk.

The channel to external devices is buffered, with transmit/receive in whole buffer blocks.

<b>Examples</b>	OPEN 1,0	To read keyboard
	OPEN 1,1,0	Read from tape
	OPEN 1,8,15, "COMMAND"	Send command to Disk

**Token** 176

**Name** Logical OR

**Class** Logical Operator

**Alternate** None

# OR

**Form** Value 1 OR Value 2

**Conditions** Returns the logical OR of the corresponding bits of the values according to the following rules:

$$0 \text{ OR } 0 = 0$$

$$1 \text{ OR } 0 = 1$$

$$0 \text{ OR } 1 = 1$$

$$1 \text{ OR } 1 = 1$$

Values are truncated to 16 bit 2's complement integer before the OR is done.

Values must be numeric.

**Note** Commonly used in statements of the form:

IF A = 5 OR B = 10 THEN - - -

OR



**Token 145**

**Class Statement**

**Alternate None**

# ON

**Form** ON variable-relation GOTO Line number 1, Line number 2 - - -  
ON variable-relation GOSUB Line number 1, Line number 2 - - -

**Conditions** Used to select one of several possible jump targets, as established by the value of the controlling variable.

If the variable has the value 1, the first line number is selected; if 2, the second, and so on.

If the variable value is less than 1 or greater than the number of line numbers, execution proceeds to the next statement.

The variable relation may be a variable or an expression.

**Note** One ON construct can eliminate the need for a group of IF statements.

**ON**



**Token** 194

**Class** Function

**Alternate** Pⓔ

## PEEK

**Form** PEEK (address)

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

The address is indicated by a decimal number, in the range 0 to 65535.

The byte value is returned in decimal form, in the range 0-255.

**Notes** Thorough understanding of PEEK and its companion POKE is necessary to full utilization of many features of the VIC-20 and to make use of machine-language routines in BASIC programming.

PEEK



**Token** 151

**POKE**

**Class** Statement

**Alternate** P⊙

# POKE

**Form** POKE address, value

**Conditions** Used to place a byte into memory at the specified address.

The address is indicated in decimal form and must be in the range 0-65536, but see the cautions below.

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

**Note** Full understanding of POKE and its companion PEEK is necessary for full utilization of many features of the VIC-20, and to use of machine language in BASIC programming.

**Caution** Improper POKE locations and/or values can give errors, terminate runs, or cause loss of program or data. The *Programmers' Reference Guide* should be consulted whenever you are not certain.

**Token** 185

**Name** Cursor Position

**POS**

**Class** Function

**Alternate** None

# POS

**Form** POS (any variable)

**Conditions** Returns the current cursor position, in spaces from left of screen.

The value is in decimal, from 0 to 21.

The variable is a dummy variable and is required.



**Token** 153

**Name** Print to Screen

**Class Statement**

**Alternate** ?

**PRINT**

# PRINT

**Form** PRINT print list

**Conditions** Places the contents of the print-list on the screen.

A print list may include:

- literals, enclosed in quotes and printed as given, subject to the nonprint character rules below
- numericals, printed as given
- variables, functions or expressions, the current value being printed.

Punctuation is used in a print list to establish a printing format for the several elements of the list, as follows;

- no punctuation, skip to next line;
- comma, skip at least one space, or go to next of the two columns starting at 0, 12.
- semicolon, no spacing.

Punctuation at the end of a print list can be used to establish the starting position for the following print list.

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

The function SPC ( ) is used to insert spaces in the print list; the space count may be 0-255.

The function TAB( ) sets the initial position of the following print item; the TAB count may be 0-255.

The nonprint characters for cursor movement, clear screen, name, and insert may be placed inside quotes, and will control screen position; overwriting may result.

Reverse characters are obtained by using quotes and the control and RVS ON keys. Use the control and RVS OFF keys, or the Return key to terminate.

Color for the characters is obtained by using quotes and the control key with the appropriate color key. All subsequent characters are printed in the new color.

See the *Programmers' Reference Guide* for the use of delete and other special characters, and for POKE commands for special color effects.

**Note** Normally, numerals must be truncated to eight column length (or less) to place two columns on screen, or TAB() can be used for printing at columns 0 and 11.

**Token** 152

**Name** Print to Device

**Class** I/O Statement

**Alternate** P<sup>Ⓡ</sup>#

**PRINT#**

# PRINT#

**Form** PRINT# file-number, print-list

**Conditions** Sends the contents of the print-list to the previously opened file.

The print-list rules are the same as for PRINT.

**Note** The easiest way to separate variables when writing a file to tape or disk is to set a string variable to CHR\$(13) or RETURN; insert this between all variable sets; a comma or semicolon can be used.

**Caution** The # is part of the short form.

**Error** FILE NOT OPEN

**Messages** NOT AN OUTPUT FILE

**Token** 135

**Name** Read Data list

**Class** Statement

**Alternate** R (E)

# READ

READ

**Form** READ variable list

**Conditions** Causes the values in a DATA list to be entered as the current values of the variables listed.

Values are read from the DATA list(s) in the order specified by the READ variable list.

If the length of all DATA lists is shorter than the length of READ requests, an error report occurs.

If the length of all DATA lists is longer than the length of READ requests, the extra is ignored.

See DATA

**Error Message** OUT OF DATA

**Token** 143

**Name** Remark

**Class** Statement

**Alternate** None

## REM

REM

**Form** REM any-text

**Conditions** Allows insertion of explanatory text in a program.

All text after a REM (including key words, and punctuation) is ignored during program execution.

To have correct representation of graphics in a REM during list, they must be enclosed in quotes; if not, they are listed as key words.

**Note** REM is a convenient place to store short machine language routines called by a program.

**Token** 140    **Name** Restore Data List Pointer

**Class** Statement                      **Alternate** RE (S)

# RESTORE

**Conditions** Sets a pointer to the start of the first DATA list in the program, allowing a READ of all or part of the data list to begin again.

RESTORE must on a line by itself.

See DATA, READ.

**RESTORE**

**Token** 142 **Name** Return to Calling Program

**Class** Statement **Alternate** RET<sup>Ⓟ</sup>

# RETURN

**Conditions** The required last statement of a sub-routine.

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

See GOSUB

**Warning** Any other statements on same program line will never be executed.

**Error Message** RETURN WITHOUT GOSUB

RETURN



**Token** 201

**Name** Right String Slice

**Class** String Function

**Alternate** R①\$

## RIGHT\$

**Form** RIGHT\$(string, length)

**Conditions** Returns the rightmost characters of the substring formed by starting at the right end and counting back the specified length.

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

If length is zero or is omitted the null string is returned.

See LEFT\$, MID\$.

RIGHT\$ has no meaning for numericals (but numbers are allowable string elements).

**Caution** The \$ is part of the short form.

RIGHT\$

**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 greater than zero but less than 1.

If value is negative, its magnitude is used as the seed for the random number generator, each seed value producing a different starting point in the sequence.

If value is zero, the internal clock (TI) reading is used as seed.

The sequence is seeded at random at power-up.

**Example**  $\text{INT}(\text{RND}(1) * 6) + \text{INT}(\text{RND}(1)*6) + 2$   
simulates the throw of two dice.

**Token** 138

**Class** Command

**Alternate** R(U)

# RUN

**Conditions** Used to start execution of a BASIC program.

RUN alone starts execution at the lowest line number in the program; the variables are cleared before starting execution.

RUN followed by a number will start execution from that line number, if it exists, or will give an error; the variables are cleared.

RUN followed by a variable name will clear that variable and then start execution at line 0, if it exists, or will give an error; other variables are cleared.

**Hint** Program memory space can be saved by entering initial variable values in the immediate mode, and starting execution with a GOTO line number.

**Error Message** UNDEF'D STATEMENT = line number does not exist.

RUN

Token 148

Class Command

Alternate S (A)

# SAVE

**Forms** SAVE  
SAVE "filename"  
SAVE "filename", device, command

**Conditions** If used alone, SAVE outputs the program in memory for recording on a tape; the tape must be manually positioned; overwrite of any program under the head will occur; if the record and play keys on the VIC recorder are down, starting is automatic; the program is stored twice; there is no identification.

A filename in quotes can be used to identify any output to be saved; it may be a literal, a string in quotes, a string variable, or a string expression.

A device number can be included to send the program to another device;

- 1 - Tape
- 2 - RS-232 Device
- 3 - Screen
- 4/5 - Printer
- 8 - Disk

SAVE

4-127 - Serial Bus Device, CR only  
128-255 - Serial Bus Device, CR-LF

A command may be included:

- 1 - Reload to some memory location.
- 2 - Do not read past this point (end-of-tape marker).
- 3 - Combines 1, 2.

See LOAD

**Error Message** DEVICE NOT PRESENT may indicate  
Command 2 or 3.

**Token** 180

**Name** Signum Function

**Class** Function

**Alternate** SⓄ

## SGN

**Form** SGN(value)

**Conditions** Returns the signum of value, as follows:  
-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 quantities.

**Hint** Use in the ON construction to select the appropriate routine covering negative, zero, or positive sign conditions, as in square-root problems.

SGN

**Token** 191

**Name** Sine Function

**Class** Function

**Alternate** SⓂ

# SIN

**Form** SIN(Angle)

**Conditions** Returns the sine of an angle which is given in radians.

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

SIN has no meaning for string quantities.

**Example** SIN(0.5) = .479425539

SIN



**Token** 166

**Name** Insert Spaces

**Class** Function

**Alternate** S(P)(

## SPC(

**Form** SPC(spaces)

**Conditions** Use in a print list to insert the specified number of spaces before the next print element.

Spaces may be specified by a number, a variable, or an expression, in integer or floating point form, which must evaluate to the range 0-255.

**Cautions** The short form includes the first parenthesis.

The following punctuation also affects the number of spaces.

SPC(  




**Token** 186

**Name** Square Root

**Class** Function

**Alternate** S $\text{\textcircled{Q}}$

## 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 quantities.

**Example** SQR(2) = 1.41421356

SQR



**Token** 169

**Name** Step Size

**Class** Statement

**Alternate** ST (E)

## STEP

**Form** FOR variable = initial value 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.

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

The increment may be numericals,  
variables or expressions, in integer or  
floating point form.

Noninteger step values are allowed.

See FOR.

STEP



**Token** 144

**Name** Stop Execution

**Class** Statement

**Alternate** SⓅ

# STOP

**Conditions** Halts program execution and returns control to keyboard.

CONT causes resumption of execution at the following statement.

Is identical to END except in the error message generated.

Is identical to keyboard RUN STOP in that execution is halted at the end of current statement processing.

**Error Message** BREAK IN may indicate either STOP or RUN STOP.

**STOP**



**Token** 196

**Name** Convert to String

**Class** String Function

**Alternate** ST<sup>®</sup>\$

## STR\$

**Form** STR\$(value)

**Conditions** Converts a value to a string.

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

A blank space is included at the beginning of the resulting string if the value is positive. This space contains a minus sign for negative values.

**Note** After conversion, the resulting quantity must be manipulated by string rules.

**Example** STR\$(2 × 6) = "12"

**Caution** The \$ is part of the short form.

STR\$



**Token** 158 **Name** Go Sub 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.

Machine language processing continues until code 60h (RTS, or Return from Sub-routine) is encountered (subject to sub-call rules); program execution resumes with the next BASIC statement.

See USR.

**Note** This statement is called SYS(tem) because it is a convenient way to use the many CBM routines available. It may also be used to call special routines; for example, one residing in a REM statement; a good understanding of machine language is necessary for full use of the computer.

SYS



TAB(

Token 163

Name Tabular Set

Class Function

Alternate T(A)(

# TAB(

**Form** TAB(column)

**Conditions** Places the print position in the specified screen column.

TAB( must be used only following PRINT or in a print list.

If current print position is beyond the column specified, the print position does not move.

TAB(0) is the leftmost position; the rightmost is the screen width minus one (usually 21, but may be set by POKE 36866).

Column may be a number, a variable or an expression, and must evaluate to the range 0-255; floating point values are rounded down.

When used in a print-list, the rules of the punctuation following apply.

**Caution** The short form, T(A) (includes the initial parenthesis.

**Error Messages** SYNTAX: PRINT is necessary. The initial parenthesis is part of the short form.

**Token** 192

**Name** Tangent

**TAN**

**Class** Function

**Alternate** None

# TAN

**Form** TAN(angle)

**Conditions** Returns the tangent of the angle which is in radians.

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

TAN has no meaning for string quantities.

**Error Message** DIVISION BY ZERO occurs if the angle is  $\pi/2$  or odd multiples thereof.



**Token** 167

**Class** Statement

**Alternate** None

**THEN**

# THEN

**Forms** 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, i.e., non-zero.

The action may be any executable BASIC command or statement; for the special case of jump to a line number, only the line number is required, the GOTO being optional; 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 *line*; multiple statements on an IF-THEN line are a frequent source of errors

**Token 164**

**Class Statement**

**Alternate None**

**TO**

# TO

**Form** FOR VARIABLE = initial value 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.

**Token** 183

**Name** USer Routine

**Class** Function

**Alternate** U<sup>Ⓢ</sup>

# USR

**USR**

**Form** USR(value)

**Conditions** Calls the user's machine language routine whose starting address is pointed to by memory locations 0001 and 0002.

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

During the routine, a value can be stored at 0001 and 0002, and recovered as the return of the USR function.

The machine language routine must end with RTS (code 60h), i.e., Return from Subroutine.

**Note** Full understanding of machine code is needed for use of USR and SYS. See the *Programmers' Reference Guide*.

**Token** 197

**Name** Numerical Value

**Class** Function

**Alternate** V(A)

# VAL

**Form** VAL (string-quantity)

VAL

**Conditions** Returns the numerical value of the digits in a string-quantity.

If the first character of the string quantity is not +, -, a decimal point, or a digit, the return is a zero.

With a numerical first character, succeeding digits or a decimal point are returned until the first nondigit character is encountered. Only one decimal point is returned.

**Token** 149

**Class** Command

**Alternate** V(E)

# VERIFY

**Form** VERIFY "Filename", device  
VERIFY

**Conditions** Checks the program "filename" on tape or disk against the program in memory.

If name and device are omitted, the test is the first program on tape; if only the device is omitted, the named program on the tape is tested. Name can be a string in quotes, or a string variable.

**Note** Verify is normally used to check for a good entry after a SAVE.

VERIFY is used to position tape for a new program recording. Use the command VERIFY "Last program name; when the VERIFY ERROR is reported, the tape is stopped beyond the end of the last program, and it is safe to record the additional program.

**Error Messages** OK - programs match.

VERIFY ERROR - programs do not match.

VERIFY

Token 146

Class Statement

Alternate W(A)

## WAIT

**Form** WAIT memory location, MASK 1, MASK 2

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

MASK 1 specifies bits to be tested, by AND with the memory location contents.

MASK 2 specifies the bit pattern to be accepted as termination, essentially using XOR with the accepted bits (any bit being tested for zero should be set to one in MASK 2).

MASK 2 may be omitted.

**Note** Normally used for I/O timing, but usable for interrupt service routines, process control, and power-down techniques. WAIT is not needed in simple applications.

WAIT

# Appendix A

## Reserved Variables

---

The following variable means names are reserved for system use. They may be used in programs, subject to the following limitations:

### **ST or SStatus**

An RS-232 associated word used to indicate a break, various errors, and the state of the 255 byte receiver buffer. Stored at memory location 144(0090h).

**Warning:** Opening an RS-232 channel can cause lose of program or data without warning. See the *Programmers' Reference Guide*.

### **TI or TIme**

A 3-byte register at memory locations 160-

162, which is used to hold the time in tenths of seconds since the last reset (by TI\$). Stored at memory location 160. TI returns the reading (up to 24 Bits, in decimal). The command form has no meaning.

### **TI\$ or Time string**

A translation of the TI register. In read form, TI\$ returns the time in HHMMSS format. In command form (TI\$ = "HHMMSS"), sets the TI variable to the time specified.

**Note:** These registers are also called a "Jiffy Clock". Do not confuse this with the timers on the interface adapter chip, addresses 9114-9119.



## Appendix B

### Operators

---

: Arithmetic, in priority order

- ↑, Raise to power (exponentiation)
- \*, /, Multiplication, Division
- +, −, Addition, Subtraction

: Arithmetical Logic

- = is equal to
- < is less than
- > is greater than
- <= or =< is less than or equal to
- >= or => is greater than or equal to
- >< or <> is not equal to.

: Logical

- AND (See key word)
- OR (See key word)
- NOT (See key word)



# Appendix C

## VIC-20 Codes

		Screen				
C	S	S	A	K		
O	E	E	S	E		
D	T	T	C	Y		
E	1	2	I		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	<u>DSC</u>	←		
9	I	i	<u>ESC</u>	W		
10	J	j		R		
11	K	k		Y		
12	L	l		l		
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		

See Programmers Manual for exact forms of diagonal entries

Screen

C	S	S	A	K
O	E	E	S	E
D	T	T	C	Y
E	1	2	I	

Token

21	U	u		L	
22	V	v		;	
23	W	w		← CSR →	
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	

























See Programmers Manual for exact forms of diagonal entries

Screen

C S S A K  
O E E S C E  
D T T I C I Y  
E 1 2 I I

Token

45	-		-	:	
46	.		.	=	
47	/		/	F3	
48	0		0	Q	
49	1		1	E	
50	2		2	T	
51	3		3	U	
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		

C O D E	Screen			K E Y	Token
	S E T 1	S E T 2	A S C I I		
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		
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					

See Programmers Manual for exact forms of diagonal entries

Screen

C	S	S	A	K
O	E	E	S	E
D	T	T	C	Y
E	1	2	I	

Token

92			£		
93					
94			↑		
95			←		
96	SP				
97					
98					
99					
100					
101					
102					
103					
104					
105					
106					
107					
108					
109					
110					
111					
112					
113					
114					
115					

Screen

C S S A K  
O E E S E Y  
D T T S C I  
E 1 2 I

Token

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 #
133	E	e			INPUT
134	F	f			DIM
135	G	g			READ
136	H	h			LET
137	I	i			GOTO
138	J	j			RUN



Indicates character/background are reversed



Screen

C S S A K  
 O T T S C E  
 D T T I I  
 E 1 2

























Token


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 OF		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
161	!				GET
162	/				NEW

Screen

C	S	S	A	K
O	E	E	S	E
D	T	T	C	Y
E	1	2	I	

Token

163	#			TAB(
164	\$			TO
165	%			FN
166	&			SPC(
167	/			THEN
168	(			NOT
169	)			STEP
170	+			+
171	-			-
172	*			*
173	/			/
174	1			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

 Indicates character/background are reversed

Screen

C	S	S	A	K
O	E	E	S	E
D	T	T	C	Y
E	1	2	I	

Token

187					RND
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		Q			
208		P			
209		Q			
210		R			

Screen

C S S A K  
O S E E A S  
D T T T C I  
E 1 2 I I  
Y

Token

211		S			
212		T			
213		U			
214		V			
215		W			
216		X			
217		Y			
218		Z			
219					
220					
221					
222					
223					
224	SP				
225					
226					
227					
228					
229					
230					
231					
232					
233					

Indicates character/background are reversed

Screen

C S S A K  
O E E S C E  
D T T E S I Y  
E 1 2 I

Token

234					
235					
236					
237					
238					
239					
240					
241					
242					
243					
244					
245					
246					
247					
248					
249					
250					
251					
252					
253					
254					
255			π		



# Appendix D

## VIC-20 Memory Map

Decimal	Boundaries	Hex
0		0
144	BASIC System	0090
256	KERNAL System	0100
512	stack      RAM	0200
828	system    1 K	033C
1024	tape buffer	0400
	3 K Expansion RAM	
4096		1000
7680	BASIC Program	
	screen      RAM	1E00
	4 K	
8192		2000
	8 K Expansion RAM	
16384		4000
	8 K Expansion RAM	
24576		6000
	8 K Expansion RAM	
32768		8000
	Character ROM	
36064	4 K	8C00
37136		9000
	I/O - 0	
	Video interface chip	
37888		9400
	Color RAM	
38912		9800
	I/O - 2	
39936		9C00
	I/O-3	
40960		A000
	8 K Expansion RAM	
49152		C000
	BASIC in ROM	
57344		E000
	Kernal in ROM	
65535		FFFF





# Appendix E

## VIC-20

### Important Memory Locations

---

Start	End	Decimal	Use
002B	002C	43-44	Points to Start of BASIC
002D	002E	45-46	Points to Start of Variables
002F	0030	47-48	Points to Start of Arrays
0031	0032	49-50	Points to End of Arrays
0033	0034	51-52	Points to String Storage
0037	0038	55-56	Points to Limit of Memory
0039	003A	57-58	Current BASIC line No.
003B	003C	59-60	Previous BASIC line No.
0045	0046	69-70	Current Variable Name
0047	0048	71-72	Current Variable Address
008B	008F	139-143	RND Seed Value
0098		152	Number of Open Files
00A0	00A2	160-162	"Jiffy" Clock
00B2	00B3	178-179	Points to Start of Tape
00C5		197	Buffer
028A		650	Current Key Pressed
			Key Repeat (128 = repeat all keys)
033C	03FB	828-1019	Cassette Buffer
1E00	1FFF	7680-8191	Screen Memory (unexpected)
8000	8FFF	32768-863	Character Generator ROM
9000	900F	36864-79	Vic Chip Registers

*See the Programmers' Reference Manual*



# Appendix F

## VIC-20

### Screen Memory Addresses

Sum these for Character Addresses

Sum these for Color Addresses

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21			
7680																								38400	
7702																									38422
7724																									38444
7746																									38466
7768																									38488
7790																									38510
7812																									38532
7834																									38554
7856																									38576
7878																									38598
7900																									38620
7922																									38642
7944																									38664
7966																									38686
7988																									38708
8010																									38730
8032																									38752
8054																									38774
8076																									38796
8098																									38818
8120																									38840
8142																									38862
8164																									38884

Example: POKE 7933,115 : POKE 38653,2 places a red heart at screen center



# Appendix G

## VIC-20 Colors

---

**Poke 36879. For This Combination**

Code	Screen		Border							
	Color		BLK	WHI	RED	CYAN	PUR	GRN	BLU	YEL
0	Black		8	9	10	11	12	13	14	15
1	White		24	25	26	27	28	29	30	31
2	Red		40	41	42	43	44	45	46	47
3	Cyan		56	57	58	59	60	61	62	63
4	Purple		72	73	74	75	76	77	78	79
5	Green		88	89	90	91	92	93	94	95
6	Blue		104	105	106	107	108	109	110	111
7	Yellow		120	121	122	123	124	125	126	127

Screen		BLK	WHI	RED	CYAN	PUR	GRN	BLU	YEL
Code	Color								
8	Orange	136	137	138	139	140	141	142	143
9	Lt. Orange	152	153	154	155	156	157	158	159
10	Pink	168	169	170	171	172	173	174	175
11	Lt. Cyan	184	185	186	187	188	189	190	191
12	Lt. Purple	200	201	202	203	204	205	206	207
13	Lt. Green	216	217	218	219	220	221	222	223
14	Lt. Blue	232	233	234	235	236	237	238	239
15	Lt. Yellow	248	249	250	251	252	253	254	254
Poke 38400 for this Graphic Color									

← These screen colors not available for characters →



# Appendix H

## VIC-20 Tone Values

Note: Notes shown are approximate, you may wish to use different values for some notes.

Note	Poke-Value
C	135
C#	143
D	147
D#	151
E	159
F	163
F#	167
G	175
G#	179
A	183
A#	187
B	191
C	195
C#	199
D	201
D#	203
E	207
F	209
F#	212

Note	Poke-Value
G	215
G#	217
A	219
A#	221
B	223
C	225
C#	227
D	228
D#	229
E	231
F	232
F#	233
G	235
G#	236
A	237
A#	238
B	239
C	240
C#	241

POKE 36878, X 0 = > x < 15 Set volume  
(15 maximum)

POKE 36874, poke-value Sets low pitch

POKE 36875, poke-value Sets middle pitch  
(135 gives approximately middle C)

POKE 36876, poke-value Sets high pitch

POKE 36877, poke-value Sets "noise"

poke value range 128-255





# Appendix I

## Hex-Decimal Converter

---

First Hex Digit	Second Hex Digit															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255



# Appendix J

## Relative Jump-Count Converter Decimal-Hex

Minus or Back Jump		First Hex									
		F	E	D	C	B	A	9	8		
-		0	16	32	48	64	80	96	112		0
F		1	17	33	49	65	81	97	113		1
E		2	18	34	50	66	82	98	114		2
D		3	19	35	51	67	83	99	115		3
C		4	20	36	52	68	84	100	116		4
B		5	21	37	53	69	85	101	117		5
A		6	22	38	54	70	86	102	118		6
9		7	23	39	55	71	87	103	119		7
8		8	24	40	56	72	88	104	120		8
7		9	25	41	57	73	89	105	121		9
6		10	26	42	58	74	90	106	122		A
5		11	27	43	59	75	91	107	123		B
4		12	28	44	60	76	92	108	124		C
3		13	29	45	61	77	93	109	125		D
2		14	30	46	62	78	94	110	126		E
1		15	31	47	63	79	95	111	127		F
0		16	32	48	64	80	96	112	128		+
		0	1	2	3	4	5	6	7		
		First Hex									
										Plus or Forward Jump	



## Appendix K

### 6501-6505 Operation Codes

Instruction	Short Description	Addressing Mode											
		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	71				
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

Addressing Mode												
Instruction	Short Description	Immediate	Zero Page	Zero Page, X	Absolute	Absolute, X	Absolute, Y	(Indirect, X)	(Indirect, Y)	Accumulator	Implied	Relative
BNE	Branch $\neq$ 0											D0
BPL	Branch +											10
BRK	Break										00	
BVC	Branch $\overline{O}$ '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			

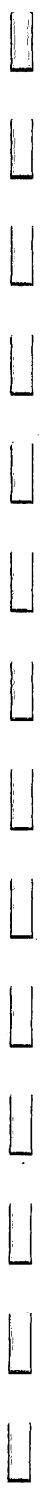












# Computer Companion for the VIC-20®

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 VIC-20 and other Commodore micros that use CBM BASIC 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 VIC-20 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 ever 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 VIC-20—or other Commodore model using CBM BASIC version 2.0—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. A well-known author, he has written several best selling computer books for TAB.

## OTHER POPULAR TAB BOOKS OF INTEREST

**Commodore 64 Graphics and Sound Programming**  
(No. 1640—\$14.95 paper; \$21.95 hard)

**Mastering the VIC-20®** (No. 1612—\$9.95 paper;  
\$15.95 hard)

**TAB TAB BOOKS Inc.**

Blue Ridge Summit, Pa. 17214

Send for FREE TAB Catalog describing over 750 current titles in print.

FPT > \$10.25

ISBN 0-8306-0613-0