TEXAS INSTRUMENTS

TI 99/4A BASIC

QUICK REFERENCE GUIDE

by Gilbert Held

WILEY QUICK REFERENCE GUIDES 2.95

REFERENCE GUIDE NOTATIONS AND FORMAT CONVENTIONS

A standard scheme for presenting the general format of BASIC language statements is employed in this reference guide. The capitalization, punctuation and other conventions are listed below:

[ ] Brackets indicate that the enclosed items are optional. Brackets do not appear in the actual statements.

{ } Braces indicate that a choice of one of the enclosed items is to be made. Braces do not appear in the actual statements.

. . . Ellipses indicate that the preceding item may be repeated. Ellipses do not appear in the actual statements.

Italics Italics indicate generic terms. The programmer must supply the actual value or wording required. See Generic Terms and Abbreviations.

Line number A line number is implied for all BASIC language statements in program mode.

Punctuation All punctuation characters, including commas, semicolons, colons, quotation marks and parentheses, must appear as indicated.

UPPERCASE Uppercase letters and words must appear exactly as indicated.

BASIC PROGRAMMING MODES

IMMEDIATE—statement(s) entered without a line number will be immediately executed by BASIC.

PROGRAM—statement(s) entered with line numbers will be executed by the RUN command.

BASIC STATEMENT FORMATS

Maximum line length is 112 characters on 4 physical lines of 28 characters per line to include the line number.

Immediate Mode format: *statement*
Program Mode format: *line statement*

GENERIC TERMS, ABBREVIATIONS AND DEFINITIONS

accessory device—Equipment such as a cassette unit, disk or printer that attaches to the computer and extends it functionality and capability.

array—A set of variables that has the same name and that is distinguished by a number known as a subscript written in parenthesis after the name. Arrays can have up to 3 subscripts in TI BASIC.

ASCII—The American Standard Code for Information Interchange whereby unique codes are used to represent letters, numbers and special symbols.

background color-code—A number between 1 and 16 that represents the color of the block or square in which the character is displayed.

character-code—The ASCII character code that defines a character.

character-set-number—A number between 1 and 16 that represents one of 16 character code sets for use in color graphics programs.

column—The horizontal position on the screen that ranges from 1 to 32.

command—An instruction that the computer performs immediately.

color-code—A number from 1 to 16 that defines a particular color.

concatenation—The joining of two or more strings.

constant—A numeric or string value that does not change.

cursor—A flashing, rectangular symbol that indicates where on the screen the next character will appear when a key is pressed.

device—A BASIC keyword that defines a specific accessory device.

diskette-name—A name up to 10 characters long containing any characters with ASCII codes 32 through 95 except the period (.) and slash (/) that gives a diskette a name for reference.

expr—Any valid expression.

expr$—Any valid string constant, variable or expression.

exprnm—Any valid numeric constant, variable or expression.

file-name—The name of any accessory device or a file located on an accessory device. The actual reference will depend upon the capability of the accessory device.

file-number—A number between 0 and 255 that is assigned to a particular file by the OPEN statement.

GENERIC TERMS, ABBREVIATIONS AND DEFINITIONS (Continued)

foreground color code—A number between 1 and 16 that represents the color of the character displayed on the screen.

format—The structure of the BASIC command, subprogram or statement.

Integer—A whole number without a decimal point.

key-unit—A number that designates the specific mode of the console keyboard, defining the upper-and lowercase characters, function keys and control character codes resulting from pressing each key.

line—A BASIC program line number between 1 and 32767, inclusive.

pattern-identifier—A 16 character hexadecimal string that defines an 8 by 8 grid of 64 dots.

program-name—An identifying name given to a program that can be up to 10 characters long containing any characters with ASCII codes 32 through 95 except the period (.) and slash (/).

row—The vertical position of a character on the screen that ranges from 1 to 24.

statement—A BASIC language statement.

status-variable—A variable whose value indicates the activity of the keyboard.

sub—Subscript. An element of an array.

subprogram—A routine built into the computer that is accessed by CALLing it by name and supplying a set of specifications that will define its operation.

var—A numeric or string variable.

var$—A string variable.

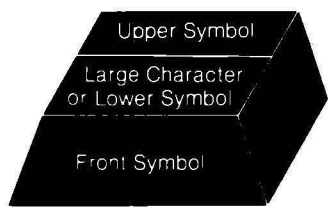
varnm—A numeric variable name.

x-return—An integer value that specifies the x coordinate of the Wired Remote Controller (joystick).

y-return—An integer value that specifies the y coordinate of the Wired Remote Controller (joystick).

___Underline (___) denotes command that can be used as a statement or a statement that can be used as a command.

DISPLAY CONTROL



KEY UTILIZATION

Alphabet keys cause alphabetical symbols to be entered into the computer when typed.

Numeric keys cause numbers to be entered into the computer when typed.

Punctuation and Symbol keys cause symbols to be entered into the computer when typed by themselves or in conjunction with other keys.

Shift and Alphabetic key cause cause uppercase alphabetic character to be entered into computer.

ALPHA LOCK key locks all the alphabetical keys into their uppercase mode.

Front Symbols on key are entered by holding FCTN (function) key down and pressing the desired key.

ENTER causes the computer to accept the information you have finished typing.

FUNCTION KEY COMBINATIONS

Holding down the FCTN key while pressing the second key activates special functions as indicated below.

Function Name	Key Combination		Operational Result
QUIT	FCTN	=	Quit BASIC and return to master computer title screen.
LEFT arrow	FCTN	S	Moves cursor to the left one position.
RIGHT arrow	FCTN	D	Moves cursor to the right one position.
UP arrow	FCTN	E	In EDIT mode enters all changes you made to current line and displays next lower numbered line for editing.
DOWN arrow	FCTN	X	In EDIT mode enters all changes you made to current line and displays next higher numbered line for editing.

FUNCTION KEY COMBINATIONS (continued)

Function Name	Key Combination	Operational Result
DELete	FCTN 1	Deletes a letter, number or other character from the lines you type.
INSert	FCTN 2	Inserts any of the letters, numbers or characters on the key into the lines you type. To stop inserting characters press ENTER, FCTN =, FCTN S, FCTN D, FCTN E or FCTN X.
ERASE	FCTN 3	Erases the line you are presently typing.
CLEAR/BREAK	FCTN 4	In EDIT mode causes current line to scroll up on screen without entering changes and results in an exit from EDIT. Otherwise, causes executing program to be suspended. Resume executing by entering CONTINUE command.

CONTROL KEY COMBINATIONS

Pressing CTRL key and appropriate letter or number key causes control character to be entered into computer. These keys are primarily used in telecommunications.

Key Combination	Mnemonic Code	Result	Key Combination	Mnemonic Code	Result
CONTROL A	SOH	Start of heading	CONTROL Q	DC1	Device control 1 (X-ON)
CONTROL B	STX	Start of text	CONTROL R	DC2	Device control 2
CONTROL C	ETX	End of text	CONTROL S	DC3	Device control 3 (X-OFF)
CONTROL D	EOT	End of transmission	CONTROL T	DC4	Device control 4
CONTROL E	ENQ	Enquiry	CONTROL U	NAK	Negative acknowledge
CONTROL F	ACK	Acknowledge	CONTROL V	SYN	Synchronous idle
CONTROL G	BEL	Bell	CONTROL W	ETB	End of transmission block
CONTROL H	BS	Backspace	CONTROL X	CAN	Cancel
CONTROL I	HT	Horizontal tabulation	CONTROL Y	EM	End of medium
CONTROL J	LF	Line feed	CONTROL Z	SUB	Substitute
CONTROL K	VT	Vertical tabulation	CONTROL .	ESC	Escape
CONTROL L	FF	Form feed	CONTROL ;	FS	File separator
CONTROL M	CR	Carriage return	CONTROL =	GS	Group separator
CONTROL N	SO	Shift out	CONTROL 8	RS	Record separator
CONTROL O	SI	Shift in	CONTROL 9	US	Unit separator
CONTROL P	DLE	Data link escape			

VARIABLE NAMING CONVENTIONS

NAME format: **F[ST]** where:

F represents the first character that must be alphabetic, an at-sign (@), a left bracket ([), a right bracket (]), a back slash (\) or a line (_).

S represents up to 14 (13 for string variables) succeeding characters that can be numeric or any of the characters allowed for the first character.

T represents the type of variable. $ for string. If type omitted, BASIC assumes variable is a floating point number.

examples:

Numeric variable names: A, X1, COST, BASE_PAY

String variable names: NAME$, X$, ADDRESS$

BASIC OPERATORS

OPERATION	OPERATOR	EXAMPLE
ARITHMETIC		
Exponentiation	^	A^B
Unary Minus	–	–A
Multiplication	*	A*B
Division	/	A/B
Addition	+	A+B
Subtraction	–	A–B
RELATIONAL		
Equal	=	A=B

OPERATION	OPERATOR	EXAMPLE
RELATIONAL (continued)		
Not equal	<>	A<>B
Less than	<	A<B
Greater than	>	A>B
Less than or equal	<=	A<=B
Greater than or equal	>=	A>=B
STRING		
Concatenation	&	A$&B$

SYSTEM COMMANDS

These commands result in the computer performing an operation at the system level and can be entered whenever the prompt and flashing cursor (> ■) appear at the bottom of the screen. The commands are normally entered without a line number; however, commands underlined in this section can also be used in a program by prefixing the command with a line number.

BREAK—Sets breakpoints at the in dicated program lines. When the program is executed, the computer will stop running the program before performing the statement on the indicated line or on the program line that contains the BREAK statement with no line number list. Program will continue execution by entering CONTINUE command.

format: BREAK[*line*[,*line*...]]

BYE—Causes all open files to be closed, erases the program in memory and all variable values and causes the master computer title screen to reappear.

format: BYE

{**CON** / **CONTINUE**}—Causes a program that suspended execution at a breakpoint to resume execution.

format: {CON / CONTINUE}

DELETE—Causes the specified program or data file to be removed from a diskette.

format: DELETE *device.program-name*

Note: The file-name and program-name are string expressions. If a string constant is used, it must be enclosed in quotes.

EDIT—Causes the computer to enter the Edit Mode and display the requested line on the screen. Changes can then be made to any character on the line other than the line number using the special keys described in the DISPLAY CONTROL section of this guide.

format: {EDIT *line* / *line* {FCTN1 / FCTN!}}

LIST—Causes the program line or lines to be displayed or printed on the specified device.

format: LIST {[*line*$_1$][*line*$_n$] / "*device-name*"[: [*line*$_1$][*line*$_n$]]}

Note: If no line numbers are entered, the entire program is displayed or printed on the specified device.

NEW—Causes the program currently in memory to be erased and the message "TI BASIC READY" to be displayed.

format: NEW

{**NUM** / **NUMBER**}—Causes the computer to automatically generate line numbers. Press ENTER after a generated line number to exit the NUMBER Mode.

format: {NUM / NUMBER} [*initial-line*] [,*increment*]

Note: If no initial-line and increment are specified, then 100 is used as the initial-line and 10 is used as the increment. If just an initial-line is specified, then 10 is used as the increment. If just an increment is specified, then 100 is used as the initial-line.

OLD—Clears the computer's memory and causes a previously SAVEd program to be copied into the computer's memory.

format: OLD *device.program-name*

{**RES** / **RESEQUENCE**}—Causes all lines in the current program to be assigned new line numbers based upon the initial-line and increment specified.

format: {RES / RESEQUENCE} [*initial-line*][,*increment*]

Note: Line number reordering result is the same as using the NUMBER command.

RUN—Causes the program stored in memory to begin execution at its lowest numbered line or at the specified line number. Prior to the program running the values of all numeric variables are set to zero and all string variables are set to null.

format: RUN[*line*]

SAVE—Causes a copy of the current program in memory to be saved on tape or disk. The SAVEd program will remain in the computer's memory after the SAVE operation.

format: SAVE {device.program-name / device.diskette-name.program-name}

TRACE—Causes the line number of each program line to be displayed before the statement is executed. TRACEing remains in effect until cancelled by the NEW command or an UNTRACE command or statement is performed.

format: TRACE

UNBREAK—Removes breakpoints from the program at the indicated lines. If no line number is specified all breakpoints are removed.

format: UNBREAK [*line*[,*line*...]]

UNTRACE—Causes the effect of a TRACE command to be cancelled.

format: UNTRACE

ISBN 0-471-88241-0
Designed by Werner Graphics

DEVICE AND PROGRAM NAMES

Device names are uppercase letters that are the predefined name of an accessory device.

Program-name specifies the particular file on a device.

Device Names	Accessory Device	Device Names	Accessory Device
CS1	cassette for input/output	DSK2	second disk
		DSK3	third disk
CS2	cassette for output only	RS232	communications or serial device
DSK1	first disk	TP	thermal printer

example: DSK1.Janrpt references the file named Janrpt on the first disk.

BASIC LANGUAGE STATEMENTS

These statements are normally entered with a line number and form a program that will be executed by entering the RUN command. Statements underlined in this section can also be entered as commands by omitting the line number. When a statement is entered as a command, it is immediately executed.

BRANCHING

GOSUB—Results in a branch to the subroutine at the indicated line number. A RETURN statement causes a branch back to the instruction following the GOSUB.

format: GOSUB *line*

example: *line* GOSUB 500

{GOTO / GO TO} —Causes an unconditional branch to the indicated line number.

format: {GOTO *line* / GO TO *line*}

example: *line* GOTO 500

IF-THEN-ELSE—Causes a branch to $line_1$ if the expression is true or a branch to $line_2$ if the condition is false. If ELSE is omitted, the next program line is executed if the condition is false.

format: IF *expr* THEN $line_1$ [ELSE $line_2$]

example: *line* IF A>B THEN 200 ELSE 300

{ON-GOSUB / ON-GO SUB} —Causes a conditional subroutine call based upon the current or computed value of the expression. The computed value must be in the range 1 to n, where n is the number of line reference numbers. A RETURN statement causes a branch back to the line number following the ON-GOSUB statement.

format: ON *exprnm* {GOSUB / GO SUB} *line*[,*line*...]

example: *line* ON X GOSUB 100,200,300

{ON-GOTO / ON-GO TO} —Causes a conditional branch based upon the current or computed value of the expression. The computed value must be in the range 1 to n, where n is the number of line reference numbers.

format: ON *exprnm* {GOTO / GO TO} *line*[,*line*...]

example: line ON X GOTO 100,200,300

RETURN—Results in a program branch to the statement immediately following the most recently executed GOSUB or ON-GOSUB statement.

format: RETURN

example: *line* RETURN

PROCESSING STATEMENTS

DATA—Creates a list of values to be assigned to variables through the use of a READ statement.

format: DATA *constant*[,*constant*...]

example: *line* DATA 1,3,5,"JOHN,S."

DEF—Statement that allows you to define your own functions to use within a program. Note that the function name cannot be used as a variable.

format: DEF {*numeric name*[(*parameter*)] = *exprnm* / *string name*[(*parameter*)] = *expr$*}

example: *line* DEF Y=A*X 2+B*X+C

DIM—Reserves space in memory for an array or matrix of variables. Space for one, two or three-dimensional arrays may be reserved.

format: DIM *var*(*sub*[,*sub*][,*sub*]) ,...

example: *line* DIM X(12),Y(15,10)

END—Terminates an executing program.

format: END

example: *line* END

FOR-TO-STEP—Initiates a loop that repeats execution of all instructions bounded by the NEXT statement until the automatically incremented variable obtains the value $exprnm_2$. If STEP clause omitted, an increment of +1 used.

format: FOR *varnm* = $exprnm_1$ TO $exprnm_2$ [STEP $exprnm_3$]

example: *line* FOR I =2 TO 10 STEP 2

LET—Assigns a value to the specified variable.

format: [LET] *var* = *expr*

example: *line* LET A=B+C

NEXT—Terminates a loop initiated by a FOR statement that has a matching variable name.

format: NEXT *varnm*

example: *line* FOR I =2 TO 4
line PRINT I ;
line NEXT I

PROCESSING STATEMENTS (continued)

OPTION BASE—Allows the lower limit of array subscripts to be set at one instead of zero.

format: OPTION BASE $\left\{\begin{matrix}0\\1\end{matrix}\right\}$

example: *line* OPTION BASE 1

RANDOMIZE—Causes a different and unpredictable sequence of random numbers to be generated by the random number function (RND) each time the program is run if used without a seed (numeric variable). If a seed is specified, the sequence of random numbers will be based upon the value of the seed.

format: RANDOMIZE [*varnm*]

example: *line* RANDOMIZE 20

READ—Assigns values from DATA statements to variables in the READ statement.

format: READ *var*[,*var*...]

example: *line* READ X,Y,P$

REM—Nonexecutable statement that allows remarks to be placed in a program.

format: REM *remark*

example: *line* REM output results

RESTORE—Causes the next READ statement values to be assigned from the first DATA statement in the program or from the DATA statement specified by a line number.

format: RESTORE [*line*]

example: *line* RESTORE 1100

STOP—Causes the program to halt execution.

format: STOP

example: *line* STOP

SCREEN INPUT/OUTPUT STATEMENTS

DISPLAY—This statement is identical to the PRINT statement when used to print items on the screen. See PRINT statement in this section.

INPUT—Optionally displays a prompt message and then accepts input data from the keyboard, assigning values to the variables listed in the statement. If no prompt message is included, a question mark (?) followed by a space will be displayed and the computer will then wait for data to be entered.

format: INPUT[*"prompt message"*:]*var*[,*var*..

example: *line* INPUT"COST":C

PRINT—Causes numbers and string values to be output to the screen.

format: PRINT [TAB(*exprnm*) $\left\{\begin{matrix}:\\;\\,\end{matrix}\right\}$]*var* $\left\{\begin{matrix}:\\;\\,\end{matrix}\right\}$ [*var*...]

where:

TAB moves the print position to the column specified.

A **comma (,)** print separator moves the beginning of the next item to be displayed to column 15 on the present line or column 1 on the next line.

A **colon (:)** print separator causes the next print item to be displayed at the beginning of the next line.

A **semicolon (;)** print separator causes adjacent print items to be displayed side by side, with no extra spaces between the values displayed.

example: *line* PRINT"SALES = ";X

FILE PROCESSING FUNCTION AND STATEMENTS

CLOSE #—Discontinues the association between a file previously OPENed and a program. If the DELETE option is used in this statement the action performed depends upon the device used. Normally DELETE will cause the file to be erased.

format: CLOSE #*file-number*[:DELETE]

example: *line* CLOSE #25

EOF—Function that can be used to determine if an end-of-file has been reached on a file stored on an accessory device.

format: EOF(*exprnm*)

The values returned are:

Value	Operational Result
0	Not end-of-file
+1	Logical end-of-file
−1	Physical end-of-file

example: *line* IF EOF(2) THEN 200

INPUT#—Statement that reads data from an accessory device and assigns the values to the specified variables in the INPUT statement.

format: INPUT #*file-number*[,REC *exprnm*][:*var*[,*var*...]]

where: The numeric expression following the keyword REC will be evaluated to designate a specific record number on a RELATIVE (random) file. If REC is omitted, the records are read sequentially.

example: *line* INPUT #13:A,B,C,D$

OPEN#—Prepares a BASIC program to use data files stored on accessory devices by providing a link between the file-number used in a program and the accessory device upon which the file is stored.

format: OPEN #*file-number*:
"device.file-name"[,*file-organization*]
[,*file-type*][,*open-mode*]
[,*record-type*][,*file-life*]

where: file-number must be 1 to 255 inclusive and cannot be the same file-number as any other file concurrently in use.

device.*file-name* refers to a diskette on which a particular file is stored.

file-organization can be specified as

SEQUENTIAL or RELATIVE. If omitted, SEQUENTIAL is assumed.

file-type designates the format of the data stored on the file as DISPLAY (ASCII printable) or INTERNAL (machine format). If omitted, the computer assumes DISPLAY format.

open-mode instructs the computer to process the file in the INPUT, OUTPUT, UPDATE or APPEND mode. If omitted, the computer assumes the UPDATE mode.

record-type specifies whether the records on the file are all the same length (FIXED) or vary in length (VARIABLE). The keyword FIXED or VARIABLE may be followed by a numeric expression that denotes the maximum length of the record. If the file is RELATIVE, the record-type must be (and defaults to) FIXED. If the file is SEQUENTIAL, the record-type defaults to VARIABLE, but may be FIXED.

file-life. Files created on the TI-99/4A are considered PERMANENT and will be so assumed if this entry is omitted.

example: *line* OPEN #2:"CS1", INTERNAL,INPUT,FIXED

PRINT#—Causes data to be written onto an accessory device.

format: PRINT *#file-number*[,REC *exprnm*][:*print-list*]

If the PRINT # statement does not end with a print-separator (comma, semicolon or colon), the record is immediately written onto the file. If the PRINT # statement ends with a print-separator, the data is held in the buffer and a "pending" print condition occurs. When a "pending" print condition occurs and the next PRINT # statement has no REC clause the data is placed in the I/0 buffer following the data already there. If the next PRINT # statement has a REC clause the pending print record will be written onto the file at the indicated position and the new PRINT # statement will be performed.

RESTORE#—Causes an open file to be repositioned at its beginning record defined by the numeric expression *exprnm*.

format: RESTORE *#file-number*[,REC *exprnm*]

example: *line* RESTORE #2

BUILT-IN SUBPROGRAMS

A special set of BASIC subprograms are built into the TI-99/4A computer that provide color graphics, sound and other capabilities. All subprograms can be used as commands and in programs.

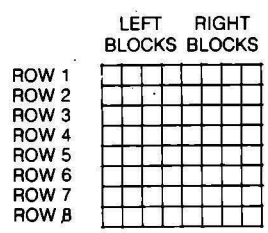
CALL CHAR—Subprogram that permits you to define your own special graphics characters as an 8-by-8 grid.

format: CALL CHAR(*character code*,"*pattern-identifier*")

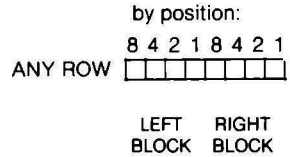
where: *character-code* specifies the code of the character to be defined.

$32 \leq char\text{-}code \leq 159$

pattern-identifier is a 16 character string expression of hexadecimal characters that defines an 8-by-8 grid as shown below.



Each row is partitioned into two blocks of four dots each whose decimal values are indicated by position:



To create a character add up the decimal values for the grid positions to be colored in each block. That decimal sum must then be turned into a hexadecimal number. The hexadecimal number is then placed in the "pattern identifier."

example: *line* CALL CHAR (50,"3C7EFFFFFFF7E3C")

Decimal to Hexadecimal Conversion

Decimal	Hexa-Decimal	Decimal	Hexa-Decimal
0	0	8	8
1	1	9	9
2	2	10	A
3	3	11	B
4	4	12	C
5	5	13	D
6	6	14	E
7	7	15	F

CALL CLEAR—Causes the entire screen to be cleared by placing the space character (ASCII 32) in all screen positions. If the space character (ASCII 32) was redefined by the CALL CHAR subprogram, then the screen will be filled with the redefined character when this subprogram is executed.

format: CALL CLEAR

example: *line* CALL CLEAR

CALL COLOR—Subroutine that assigns a foreground and background color to one of 16 character code sets.

format: CALL COLOR (*character-set-number*,*foreground-color-code*, *background-color-code*)

where:

Set#	Character Codes	Set#	Character Codes
1	32-39	9	96-103
2	40-47	10	104-111
3	48-55	11	112-119
4	56-63	12	120-127
5	64-71	13	128-135
6	72-79	14	136-143
7	80-87	15	144-151
8	88-95	16	152-159

COLOR CODE TABLE

COLOR	CODE#	COLOR	CODE#
Transparent	1	Medium Red	9
Black	2	Light Red	10
Medium Green	3	Dark Yellow	11
Light Green	4	Light Yellow	12
Dark Blue	5	Dark Green	13
Light Blue	6	Magenta	14
Dark Red	7	Gray	15
Cyan	8	White	16

Note: Default foreground-color is black, default background-color is transparent.

example: *line* CALL COLOR (2,3,16)

CALL GCHAR—Subprogram that allows a character to be read from anywhere on the display. The ASCII code of the character is placed into the numeric variable upon execution of this subprogram.

format: CALL GCHAR(*row,column, varnm)*

example: *line* CALL GCHAR(1,5,X)

CALL HCHAR—Subprogram that places the designated character at a specified location on the screen and, optionally, repeats it horizontally.

format: CALL HCHAR(*row,column, character-code*[,*number of repetitions*])

where:

$0 \le$ *character code* ≤ 32767

$0 \le$ *number of repetitions* ≤ 32767

example: *line* CALL HCHAR(5,5,65,5)

Note: The computer will convert the value specified in the character code to a number between 0 and 255.

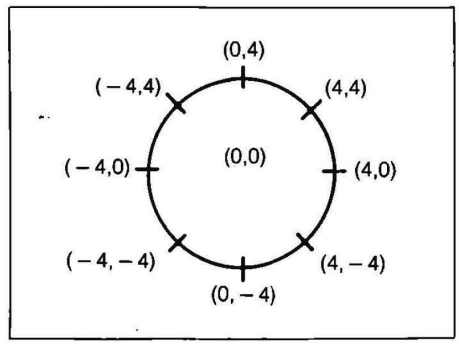
CALL JOYST—Subprogram that returns values based upon the current position of the Wired Remote Controllers (joysticks). Be sure the ALPHA LOCK is "up" or the joysticks may not function properly.

format: CALL JOYST(*key-unit,x-return, y-return*)

where: *key-unit* is a numeric expression which, when evaluated, has a value of 1 through 4 as follows:

1 = controller 1
2 = controller 2
3,4,5 = specific modes for console keyboard

x-return and *y-return* are variables whose values are assigned as follows based upon the position of the Wired Remote Controllers:



example: *line* CALL JOYST(1,X,Y)

CALL KEY—Subprogram that transfers one character at a time from the keyboard directly to a program.

format: CALL KEY(*key-unit,return-variable,status-variable*)

where: *key-unit* indicates which keyboard mode is the input device.

0 = console keyboard, in mode previously defined by CALL KEY
1 = left side of console keyboard or remote control 1.
2 = right side of console keyboard or remote control 2.
3,4,5 = specific modes for console keyboard.

return-variable is a numeric variable that will contain the character code of the key pressed.

status-variable is a numeric variable that indicates what happened at the keyboard as follows:

+1 a new key pressed since the last CALL KEY routine.
−1 the same key was pressed as was pressed during last CALL KEY routine.
0 no key was pressed.

example: *line* CALL KEY(0,KEY,STATUS)

CALL SCREEN—Subprogram that causes the entire screen background to change to the color specified by the *color-code.*

format: CALL SCREEN(*color-code*)

example: *line* CALL SCREEN(8)

CALL SOUND—Subprogram that has the computer produce sounds of a specified duration, frequency and volume.

format: CALL SOUND(*duration,frequency*$_1$, *volume*$_1$[,*frequency*$_2$,*volume*$_2$] [,*frequency*$_3$,*volume*$_3$][,*frequency*$_4$, *volume*$_4$]

where: *duration* is a numeric variable that specifies the length of the tone in milliseconds, such that

$1 \le$ *duration* ≤ 4250 and $-1 \le$ *duration* ≤ -4250

Note: 1 second equals 1000 milliseconds. If duration is positive, the computer waits until SOUND is completed prior to performing a new CALL SOUND subprogram. If duration is negative, the previous sound is stopped and

the new CALL SOUND subprogram is begun immediately.

frequency is a numeric variable that specifies the tone actually played, such that

$110 \le frequency \le 44733$ (tone)
$-1 \le frequency \le -8$ (noise)

volume is a numeric variable that specifies how loud the tone is, such that

$0 \le volume \le 30$

where 0 is the loudest and 30 the quietest.

Note: A maximum of 3 tones and 1 noise can be simultaneously activated.

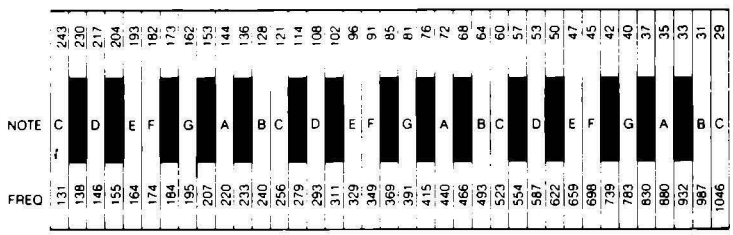
example: *line* CALL SOUND(1000,880,0)

CALL VHCAR—Subprogram that will display the specified character at the designated position on the screen and optionally repeat it vertically down the screen.

format: CALL VCHAR(*row,column,character-code*[,*number-of-repetitions*])

example: *line* CALL VCHAR(5,10,80,7)

MUSICAL TONE FREQUENCIES



frequency can also specify white or periodic noise as indicated below:

Noise Characteristics

Frequency Value	Characteristics
$-1, -2, -3$	"Periodic Noise" types 1, 2 and 3.
-4	"Periodic Noise" that varies with the frequency of the third tone specified.
$-5, -6, -7$	"White Noise" types 1, 2, and 3.
-8	"White Noise" that varies with the frequency of the third tone specified.

BASIC FUNCTIONS

FUNCTION FORMAT AND DESCRIPTION

ABS(*exprnm*)—Returns the absolute value of the argument.

ASC(*expr$*)—Returns the ASCII character code of the first character of the specified string.

ATN(*exprnm*)—Returns the arctangent of an angle of exprnm radians, where degrees * π/180 represents the equivalent angle in radians.

CHR$(*exprnm*)—Returns the character (string value) corresponding to the specified ASCII code exprnm.

COS(*exprnm*)—Returns the cosine of an angle of exprnm radians, where degrees * π/180 represents the equivalent angle in radians.

EXP(*exprnm*)—Returns the base of the natural logarithm, (2.718281), raised to the specified power.

INT(*exprnm*)—Returns the next smaller integer less than or equal to exprnm. e.g., INT(6.9) returns 6 and INT(−6.9) returns −7.

LEN(*expr$*)—Returns the length of the specified string.

LOG(*exprnm*)—Returns the natural logarithm of the specified number.

POS(*expr$$_1$*,*expr$$_2$*, *exprnm*)—Returns the first occurrence of the string *expr$$_2$* within string *expr$$_1$*, with the search beginning at position exprnm in *expr$$_1$*.

RND—Returns a pseudo-random number between 0 and 1.

SEG$(*expr$*,*exprnm*$_1$, *exprnm*$_2$)—Returns a substring of the string *expr$* starting at position *exprnm*$_1$ whose length in characters is defined by *exprnm*$_2$ where exprnm$_1 \ge 1$ and exprnm$_2 \ge 0$.

SGN(*exprnm*)—Returns +1 if *exprnm* is positive, −1 if negative and 0 if its value is zero.

SIN(*exprnm*)—Returns the sine of an angle of *exprnm* radians, where degrees * π/180 represents the equivalent angle in radians.

SQR(*exprnm*)—Returns the square root of the numeric expression where exprnm ≥ 0.

STR$(*exprnm*)—Converts the number specified by the argument into a string.

TAN(*exprnm*)—Returns the tangent of the angle of *exprnm* radians.

**VAL(*expr$*)**—Returns the numeric value of a string where expr$ is a valid representation of a numeric constant.

- Thanks to 99er User: - Dano!, for the Scan of this Document.

- Revisited by TI99 Italian User Club (info@ti99iuc.it) in March, 2014