SV·318

PERSONAL COMPUTER
BASIC QUICK REFERENCE GUIDE

SPECTRAVIDEO™

CONTENTS

Published by
SPECTRAVIDEO INTERNATIONAL LTD.

First Edition
First Printing 1983
Printed in Hong Kong

SECTION 1

SPECIAL CHARACTERS

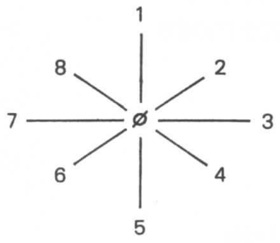
Asc	(∧ means [CTRL])	
2	**∧ B**	Move cursor to start of previous word
3	**∧ C**	break
5	**∧ E**	truncate line (clear text from current cursor postion to end of logical line)
6	**∧ F**	move cursor to start of next word
7	**∧ G**	beep
8	**∧ H**	backspace, deleting character on the left of cursor
9	**∧ I**	tab (8 spaces)
11	**∧ K**	cursor home
12	**∧ L**	clear screen
13	**∧ M**	carriage return, enters current line
14	**∧ N**	move cursor to end of line
18	**∧ R**	toggles insert/replace mode
21	**∧ U**	clear logical line
28	**∧ **	cursor right
29	**∧]**	cursor left
30	**∧ [SHIFT] [6]**	cursor up
31,	**∧ _**	cursor down
	∧ < stop >	halt program execution
	< CLR/HM >	same as ∧L, shift to home cursor
	< del >	Shift < del > same as ∧H, use as is to delete character at cursor
	< Ins >	same as ∧R
	< stop >	toggles pause or resume program execution
	&B	prefix for binary constant
	&H	prefix for hexidecimal constant
	&O	prefix for octal constant
	:	seperate statements typed on the same line
	?	same as PRINT
	'	same as REM but use more memory
	.	denote CURRENT LINE for RENUM, DELETE, LIST, LLIST, RUN commands

PROGRAMMABLE FUNCTION KEYS

Key number	**Initial definition**
< F1 >	color
< F2 >	auto < cr >
< F3 >	goto
< F4 >	list
< F5 >	run < cr >
< F6 >	color 15, 4, 4 < cr >
< F7 >	switch
< F8 >	cont < cr >
< F9 >	list
< F10 >	∧ L run < cr >

Joystick Cursor Control Pad

control cursor movement in eight directions



VARIABLE TYPE DECLARATION CHARACTERS

Variable	Range	No. of Byte
$ String	0..255 characters	3 + # of characters
% Integer	—32768..32767	2
! Single precision	7.1 digit floating integer	4
# Double precision	16.8 digit floating point	8

FORMAT NOTATION

Whenever the format of a statement or command is given, the following rules apply:

1. Items in capital must be input as shown.
2. Items in lower case letters enclosed in bracket (< >) are to be supplied by the user.
3. Items in square brackets ([ ]) are optional.
4. All punctuation except angle brackets and square brackets (i.e., commas, parentheses, semicolons, hypens, equal signs) must be incld where shown.
5. Items followed by an ellipse (...) may be repeated any numbers of time (up to the length of line).
6. "string" means a string expression.
7. "exp" means a numeric expression either constant or a variable.
8. "var" means any variable.
9. "line" and "line number" both means line number.
10. "f" is a file number or expression that evaluates to a file number.
11. "n" means an integer.
12. "x", "y" denotes X, Y co-ordinate of the screen.

SECTION 2

BASIC COMMAND

Command	Function
AUTO [< line >] [, < inc >]	generate line numbers automatically **e.g. AUTO 100, 10**
BLOAD "filename" [, < load address >]	load a machine language program into memory
BSAVE "filename", start addr, end addr [, execution addr]	Save a section of memory to "filename"
CLEAR [[< exp 1 >] [, < exp 2 >]]	exp 1 sets the amount of string space, exp 2 sets the end of memory **e.g. CLEAR 100 or CLEAR 100, & HFA00**
CLOAD < "filename" >	loads the file from cassette
COLOR [< exp 1 >] [, < exp 2 >]	exp 1, a color number for text display, exp 2 the background color. **e.g. COLOR 15, 4**
CONT	continue program execution
CSAVE < "filename" >	save current program to cassette
DELETE < startline > [— < endline >]	delete program lines

Command	Function
KEY LIST	list all the programmable function key's contents.
LIST [< line > [—[< line >]]]	11st program lines on the screen **e.g. LIST 10—50**
LLIST	same as list, except to the printer
LOAD < "filename" >	load an ASCII file **e.g. LOAD "PROGI"**
MAXFILES = < exp >	Set the maximum number of files BASIC can open during execution **e.g. MAXFILES = 3**
MERGE < "filename" >	merge a ASCII program into memory, current program's identical line numbers will be replaced. **e.g. MERGE "SECOND"**
MOTOR [ON] [OFF]	turn cassette motor on or off
NEW	delete current program and variables from memory
RENUM [[< newline >] [, < oldine >] [, < inc >]]]	renumber program lines **e.g. RENUM 1000,, 100**
RUN [< linenumber >]	run a program from (linenumber) default is first line
SAVE < "filename" >	save the program in memory with name "filename" as an ASCII file
SOUND [ON] [OFF]	turn sound of the cassette audio on or off
SWITCH	switch into another bank

Command	Function
SWITCH STOP	switch into another bank and force a CTRL-STOP
TRON	turn on trace for program execution
TROFF	turn trace off
WIDTH [39] [40]	set the display line width **e.g. WIDTH 39**

BASIC STATEMENTS

Statement	Function
BEEP	make a beep sound
CLICK [ON] [OFF]	turn on or off key click sound
DEF FNx[(<arg>)]	define an arithmetic or string function **e.g. DEF FNA(x) = ATN(x/sqr (—x*x + 1))**
DEF USR<n> = <addr>	define the entry address for the nth machine language routine **e.g. DEF USRO = &HFA01**
DEF <vartype>[<exp> [,[—<exp>]]...]	define range of variable begin with letter # exp to default their type notation where "type" is INT, DBL, SNG or STR **e.g. DEFINT I,J,M** **e.g. DEFSTR A,B—F** **e.g. DEFSTR S—Z**
DIM <var(n)> (,<var(n)> [,...])	n can be any integer, this allocate n number of elements for array variables and specify their maximum subscript values **e.g. DIM I (52), N$(4)**
END	terminate program execution, close all files and return to command level

Statements	Function
ERASE <var> [,<var>...]	release space which was used by the array name "var" **e.g. ERASE I, N$**
ERROR <n>	generate error of code n
FOR <variable> = <exp 1> to <exp 2> [STEP <exp 3>]	use with NEXT statement to repeat a sequence of program lines, variable is first set to value of exp 1 then added with exp 3 until the value of exp 2 is reached **e.g. FOR CARD = 1 to 52 STEP 13**
GOSUB <linenumber>	call a subroutine in BASIC, see RETURN
GOTO <linenumber>	branch to specified linenumber
IF <exp> THEN <statement> [[: <statement> [:...]] ELSE statement [: statement [:...]]]	If exp is not 0, the THEN clause is executed otherwise the ELSE clause line statement is executed **e.g. IF A > B THEN PRINT A ELSE PRINT B**
IF <exp> GOTO <line> [ELSE <statement> [:<statement> [.,...]]]	If exp is not 0 then the GOTO clause is executed. Otherwise the ELSE clause or next statement is executed
[LET] <var> = <exp>	assign a value to a variable

Statements	Function
MID$(< string exp >, < n > [,< m >]) = < string exp 2 >	to replace a portion of string exp 1 with string exp 2 starting in string exp 1 nth character with m number of character **e.g. MID$("steel", 4) = "a"**
NEXT < var > [,< var > [,...]]	deleimits the end of a FOR loop
ON exp GOSUB < line > [,< line > [,...]]	the subroutine that will jump to depend upon the value of the exp. and the starting address of the subroutines are indicated by the linenumbers in the GOSUB clause. In the example I must be between 1 and 3 **e.g. ON 1 GOSUB 100, 200, 300**
ON exp GOTO < line > [,< line > [,...]]	same as ON exp GOSUB, see GOTO and GOSUB **e.g. ON L GOTO 1000, 2000, 2020**
OUT < port >, < byte >	puts byte specified to output prot specified
POKE < address >, < byte >	puts byte specified into memory location specified. USE WITH EXTREME CAUTION as random poking can cause the system to CRASH!
REM [any text]	the line following the REM will not be executed, allow user to put comments in program **e.g. REM put comment here**

Statements	Function
RESTORE [< linenumber >]	reset DATA pointer so that the previous used DATA statement can be re-read
RETURN [< linenumber >]	return subroutine to statement following last GOSUB executed
STOP	stop the program exution, print BREAK message, and return to command level
SWAP < var >, < var >	exchanges value of two variables
WAIT < port >, < mask > [, < select >]	suspends program excution read input at port until (input bit XOR select AND with mask) returns non-zero
SOUND < exp 1 >, < exp 2 >	put value exp 2 into sound generator reg number exp 1 **e.g. SOUND 7, & B11101111**
SWITCH	function to return a ϕ if in BANK ϕ a 1 if in BANK 2

GRAPHIC STATEMENT AND COMMAND

Statements	**Function**
CLS	clear graphic screen
CIRCLE (<xcenter>, <ycenter>), <radius> [,<color> [,<start>, <end> [,<aspect ratio>]]]	]] draws an elipse with a center and radius as indicated by the first of its arguments, the color default is foreground color, start and end is — 2PI to 2PI, the ratio is for Horizontal and Vertical ratio of the elipse **e.g. CIRCLE (128, 96) ,,,, 1/7**
COLOR [<foreground>], [<background>], [<border>]	set the color for the 11st of arguments **e.g. COLOR 15, 4, 1**
DRAW <string> or <string var>	use to draw figure on the screen according to the graphic macro language **e.g. DRAW "M100, 100; SIOU25R25D25XA$;"**
GET [(<x1, y1>) — (<x2, y2>),] <arrayname>	to read points from the graphic screen or from defined are of the graphic screen. Array must be dimensioned large enough to hold the data **e.g. GET (1,1) — (256,5), A**

Statements	Function
LINE [(<x1, y1>)] — (<x2, y2>) [,[<color>] [,<b[f]>]]]	draws straight line connecting the two coordinate specified or if b [f] is present draws or fill rectangle **e.g. LINE (10, 10) — (100,30),, bf**
LOCATE <exp 1>, <exp 2>	position the graphic cursor to the starting coordinate pointed to by exp 1 and exp 2, can used for LINE, POINT, PRINT
PAINT <exp 1, exp 2>, paint color	fill in an arbitrary graphics figure of the specified fill color. Note: The color employed to paint an object should be the same one as the border color, otherwise, unexpected result occurs. **e.g.**

```
 5 REM — CIRCLE DRAW WITH
         COLOR —
10 SCREEN 1
20 C = 2
30 FOR I = 100 TO 1 STEP-10
40 CIRCLE (128,96), I,C
50 PAINT (128,96), C
60 C = C + 1
70 NEXT I
80 GOTO 80
```

POINT (<exp 1>, <exp 2>)	read the color of a pixel in the graphic modes **e.g. C = POINT (128,96)**

Statements	**Function**
PSET (<exp 1>, <exp 2>) [,<color>]	to draw a dot at the assigned position on the screen using the foreground color or (color) if specified **e.g. PSET (128,96), 1**
PRESET (<exp 1>, <exp 2>) [,<color>]	same as PSET except draw in background color if (color) not specified
PUT (<exp 1>, <exp 2>), <arrayname> [<operation>]	output graphic patterns in the array to assigned position on the screen, operation can be: PSET output pattern as is PRESET reverse pattern foreground/background color AND combine graphic pattern color with screen pattern OR graphic pattern overlapping the screen data. XOR perform XOR with screen data. If the matching pixel from the array and the screen are the same then that pixel will be displayed in background color else it will be displayed in the foreground colour.
PUT SPRITE <sprite planet>, (<x,y,>) [,<color>] [,<n>]	set up sprite attribute, when a field is omitted, the current value is used, see SPRITE$. **e.g. PUT SPRITE 0, (128,96) ,, 5**

Statements	Function
SCREEN <exp 1>, <exp 2>	use exp 1 to select graphic modo. 1 = high res. 2 — low res. exp 2 is to select the size of sprite (if used) see GRAPHIC CHART
SPRITE$(<n>) = <string exp>	to define a pattern as sprite, the number of string character depends on the sprite size, n must be less than 256 when size of sprite is 0 or 1 (8 bytes), less than 64 when size of sprite is 2 or 3 (32 bytes) **e.g. SPRITE$(0) = CHR$ (&B00010000) + ...** **e.g. SPRITE$(0) = CHR$(16) + ...**
SPRITE$(<n>)	function to return either a 8 byte character string or 32 byte character string of the sprite number n depend on the size of sprite selected **e.g. A$ = SPRITE$(0)**
VPEEK (<vram addr>)	return byte value from location in video ram **e.g. C = VPEEK (10)**
VPOKE <vram addr>, <byte>	put a byte into video ram address e.g. VPOKE 10,65

GRAPHIC MACRO LANGUAGE (GML)

U <n>	move up
D <n>	move DOWN
L <n>	move LEFT
R <n>	move RIGHT
E <n>	move diagonally up and right
F <n>	move diagonally down and right
G <n>	move diagonally down and right
H <n>	move diagonally up and left
<n>	denotes the distance to move times the scaling factor.
S <n>	set scaling factor <n> The scaling factor multiplied with the distances given in the U, D, L, R, E, F, G, H and M command.
B	Move but don't plot
N	Move but return to original position.
A <n>	Set the angle, <n> = ϕ; ϕ degree
	= 1; 90 degrees
	= 2; 180 degrees
	= 3; 270 degrees
C <n>	Set color number.
X <string>	Execute string, must be terminated by a;

INTERRUPT CONTROL COMMAND AND STATEMENT

Statement	Function
ON ERROR GOSUB linenumber	Defines the line number of the subroutine to execute when an error has been detected
ON INTERVAL = <exp> GOSUB linenumber	Sets the line number to exectue at every other machine interrupt cycle (60 per second) specified by <exp> **e.g. ON INTERVAL = 60 GOSUB 1000**
ON KEY [(<n>] GOSUB [<line>] [,<line>] [,...]	Specify the line number corresponding to the [line] offset n in the statement to execute when ever a function key number n has been depressed **e.g. ON KEY GOSUB 100, 200, 300**
ON STOP GOSUB <linenumber>	Define jump address when a CTRL-STOP key is pressed
ON SPRITE GOSUB <linenumber>	Define jump address when sprites collision occurs

Statements	Function
ON STRIG GOSUB <linenumber>	Define the starting line of the subroutine employed when any of the Joystick button n = (ϕ) — SPACE BAR, (1) — JOYSTICK 1,(2) — JOYSTICK 2 or <space bar> is depressed All the ON <interrupt type> GOSUB statement acts much like a GOSUB statement except the execution of these statement are interrupt driven
INTERVAL ON/OFF/STOP	To enable, disable, or terminate the BASIC timer interrupt trapping
KEY [n] ON/OFF/STOP	To enable, disable, or terminate interupts cause by a function key
STOP ON/OFF/STOP	To enable, disable, or terminate CTRL-STOP trapping
STRIG ON/OFF/STOP	To enable, disable, or terminate Joystick button or space bar trapping
SPRITE ON/OFF/STOP	To enable, disable, or terminate the interrupt generated when two or more sprites collide

SOUND COMMAND AND STATEMENTS

Statement	Function
SOUND <psg reg>, <byte>	
	put byte into the psg register, range from 0 to 13
	e.g. SOUND 13, 0

PSG / register	Function
0, 1	Tone of channel A
2, 3	Tone of channel B
4, 5	Tone of channel C
6	Noise generator
7	MIXER Format:

B7	B6	B5	B4	B3	B2	B1	Bϕ
NOT USED		NOISE ENABLE			TONE ENABLE		
		C	B	A	C	B	A

8	Amplitude control of channel A bit O—3, fixed amplitude control level bit 4, AMPLITUDE MODE.
9	Amplitude control of channel B
10	Amplitude control of channel C
11, 12	Envelop period control range 0..65535 EP : 1789772.5 x fsec / 1536 Envelope cycle/shape control
13	SHAPE:

0 — 3,9	4 — 7, 15
8	10
11	12
13	14

PLAY string exp

to play melody indicated by a character string which syntax is described in the Music Macro Language

e.g. PLAY "CDEFGABL = A; O = OCT; CDEFGAB"

e.g. PLAY A$

MUSIC MACRO LANGUAGE

A—G [#][+][−] L <n> M <exp>

Play the note, A "#" or "+" means sharp, and "—" means flat Set the length of each note, n maybe ranged from 1 to 64
Set the tone period to exp, <exp> is the period of occurance set by the Sn command, this command is disabled if the V command is used

Statements	**Function**
N <n>	Play note n, n (O — 84) range in 7 octives, n = O means rest
O <n>	Octave — set the octave of the notes to be played. Default is 4
R <n>	Rest n period range 1 — 64. Rest is the same as Ln
S <n>	Set the shape of noise output, n can be 0 — 15, refer to chart below
T <n>	Tempo — sets the number of L4's in a second. n range is 32 to 255. Default is 120
V <n>	Volumn — sets the volumn of the tone generated. n is in the range of : : : Dot or period. After each note causes the note to play 3/2 times the period determined by L (length) times T (tempo). Multiple dots may appear after the note
X	Execute substring

SECTION 3

OPERATORS

Symbol	Function
=	Assignment or equallty test
—	Negation or subtraction
+	Addition or string concatenation
.	Multiplication
/	Division (floating point result)
↑	Exponantiation
\	Integer division (integer result)
MOD	Integer modulus (integer result)
NOT	One's complement (integer)
AND	Bitwise AND (integer)
OR	Bitwise OR (integer)
XOR	Bitwise exclusive OR (integer)
EQV	Bitwise equivalence (integer)
IMP	Bitwise implication (integer)
=, <.>, < =, = <, > =, = > < >	Relational tests (result is TRUE = — 1 or FALSE = 0)

The precedence of operators is:

(1) Expressions in parentheses
(2) Exponentiation (A ↑ B)
(3) Negation (—X)
(4) *,/
(5) \
(6) MOD
(7) + , —
(8) Relational operators (= , < >, <.>, < = , > =)
(9) NOT
(10) AND
(11) OR
(12) XOR
(13) IMP
(14) EQV

ARITHMETIC FUNCTIONS

Function	Action	Example
ABS (exp)	Absolute value of expression	Y = ABS(A + B)
ATN (exp)	Arctangent of the expression (in radians)	PRINT ATN(A)
COBL (exp)	Convert the expression to a double precision number	A = CDBL(Y)
CINT (exp)	Convert the expression to an integer	B = CINT(B)
COS (exp)	Cosine of the expression (in radians)	A = COS(2.3)
CSNG (exp)	Convert the expression to a single precision number	C = CSNG(X)
EXP (exp)	Raises the constant e to the power of expression	B = EXP(C)
FIX (exp)	Returns truncated integer of expression	J = FIX(A/B)
FRE (exp)	Gives memory free space not used by BASIC	PRINT FRE(0)
INT (exp)	Evaluates the expression for the largest integer calculated	C = INT(X + 3)
LOG (exp)	Gives the natural logarithm of the expression	D = LOG(Y—2)

RND [(exp)]	Generates a random number.
	Expression:
	< 0 seed newsequence
	= 0 return previous random number
	> 0 or ommitled, return new random number
		E = RND(1)
		N = RND(—TIME)

Function	Action	Example
SGN (exp)	1 if expression > = 0 if expression = 1 — 1 if expression 0	**B = SGN(X + Y)**
SIN (exp)	Sine of the expression (in radians)	**B = SIN(A)**
SQR (exp)	Square root of expression	**C = SQR(D)**
TAN (exp)	Tangent of the expression (in radians)	**D = TAN(3.14)**

STRING FUNCTIOINS

Function	Action	Example
ASC (string)	Returns the ASCII value of the first character of string	**PRINT ASC(A$)**
CHR$ (exp)	Returns a one-character string whose character has the ASCII code of exp	**PRINT CHR$(48)**
FRE (string)	Returns remaining memory free space	**PRINT FRE(A$)**
HEX$ (exp)	Converts a number to a hexadecimal string	**H$ = HEX$(100)**
INKEY$	Returns all the one-character string read from terminal or null string if no character pending at terminal.	**A$ = INKEY$**
INPUT$ (length [,[#] m]	Returns a string of length characters read from console or from a file. Characters are not echoed.	**X$ = INPUT$(4)** **X$ = INPUT** **X$ = INPUT$(5, #2)**
INSTR [[exp], string 1, string 2]	Returns the first position of the first occurrence of string 2 in string 1 starting at position exp	**INSTR(A$,":")** **INSTR(3, X$, Y$)**
LEFT$ (string, length)	Returns leftmost length characters of the string expression	**B$ = LEFT$(X$,8)**

Function	Action	Example
LEN (string)	Returns the length of a string	**PRINT LEN(B$)**
MID$ (string, start [, length])	Returns characters from the middle of the string starting at the position specified to the end of the string or for length characters	**A$ = MID$(X$,5,10)**
OCT$ (exp)	Converts a number to an octal string	**O$ = OCT$(100)**
RIGHT$ (string, length)	Returns rightmost length characters of the string expression	**C$ = RIGHT$(X$,8)**
SPACE$ (exp)	Returns a string of exp spaces	**S$ = SPACE$(20)**
STR$ (exp)	Converts a numeric expression to a string	**PRINT STR$(35)**
STRING$ (length, string)	Returns a string length long containing first character of string	**X$ = STRING$ (100, "A")**
STRING$ (length, exp)	Returns a string length long containing characters with numeric value exp	**Y$ = STRING$ (100, 42)**
VAL (string)	Converts the string representation of a number to its numeric value	**PRINT VAL ("3.1")**

I/O AND SPECIAL FUNCTIOINS

Function	Action	Example
CVI (string)	Converts a 2-character	**Y1 = CVS(N$)**
CVS (string)	Integer (CVI). Converts a 4-character	**A% = CVI(B$)**
CVD (string)	String to a single precision number (CVS). Converts an 8-character string to a double precision number (CVD).	**C # = CVD(X$)**
ERL	Error line number	**PRINT ERL**
ERR	Error code number	**IF ERR = 60 Then...**
INP (port)	Inputs a byte from an input port	**PRINT INP(21)**
LPOS (n)	Returns carriage position of line printer (n is dummy argument)	**IF LPOS(3) > 60...**
MKI$ (value)	Converts an integer to a 2-character	**D$ = MKS$(A)**
MKS$ (value)	String (MKI$). Converts a single	**A$ = MKI$(B%)**
MDK$ (value)	Precision value to a 4-character string (MKS$). Converts a double precision value to an 8-character string (MKD$).	

Function	Action	Example
PEEK (exp)	Reads a byte from memory location specified by expression	PRINT PEEK (H2000)
POS (n)	Returns carriage position of terminal (n is dummy argument)	IF POS(3) > 60...
SPC (exp)	Used in PRINT statements to print spaces	PRINT SPC(5), A$
TAB (exp)	Used in PRINT statements to tab carriage to specified position	PRINT TAB(20), A$
USR [n] (arg)	Calls the user's machine language subroutine with the specified argument. See DEF USR.	X = USR2(Y)
VARPTR (var)	Returns address of variable in memory or zero if variable has not been assigned a value	I = VARPTR(X)
ATN (exp)		

PRINT USING FORMAT FIELD SPECIFIERS

NUMERIC

Specifier	Possible Digits	Field Characters	Definition	Example
#	1	1	Numeric field	####
.	0	1	Decimal point	#.#
+	0	1	Print leading or trailing sign. Positive numbers will have "+", negative numbers will have "–"	+## ###+
–	0	1	Trailing sign. Prints "–" if negative, otherwise blank.	##.##–
**	2	2	Leading asterisk	**###.##
$$	1	2	Floating dollar sign. $ is placed in front of the leading digit.	$$##.##
**$	2	3	Asterisk and floating dollar sign.	**$#.##
	1	1	Use comma every three digits (left of decimal point only.)	##.###.##

↑↑↑↑	0	4	Exponential format. Number is aligned so leading digit is non-zero.	#.###↑↑↑↑
underscore	0	1	Next character literal	_!#.#

STRING

!	Single character	!
\<spaces>\	2 + number of spaces character field	\ \
&	Variable length field	&

INPUT/OUTPUT STATEMENTS

Statement	Syntax/Function	Example
CLOSE	CLOSE [[#]] [,[#] [...]] Closes files, if no argument, all open files are closed.	**CLOSE 6**
DATA	DATA list of constants Lists data to be used in a READ statement.	**DATA 2.3, "PLUS", 4**