4 of 1

2/5/79

A T A R I B A S I C (S H E P A R D S O N ' S)

TABLE OF CONTENTS Rev 3 4./10./79

SUMMARY OF STANDARD STATEMENTS, FUNCTIONS, ETC. IN 8K BASIC

ABS ( )
ADR ( )
AND
ASC ( )
ATN ( )

BYE

CLOAD
CHR $ ( )
CLOSE
CLR
COLOR
CONT
COS ( )
CSAVE

ENTER

DATA

DEG
DIM
DOS
DRAWTO

END
EXP ( )

FOR
FRE ( )

GET

GOSUB
GOTO
GRAPHICS

IF

INPUT
INT ( )

LEN ( )

LET

LIST

LOAD
LOCATE
LOG ( )
LPRINT

NEW

NEXT
NOT
NOTE

ON
OPEN
OR
PADDLE ( )

PEEK ( )
PLOT
POINT

POKE
POP
POSITION
PRINT
PTRIG

PUT

RAD

READ
REM
RESTORE
RETURN

RND ( )

RUN

SAVE

SETCOLOR

SGN ( )
SIN ( )
SOUND

SQR ( )
STATUS

STEP
STICK
STRIG
STOP
STR $ ( )

THEN
TO
TRAP

USR ( )

VAL ( )

XIO

General Notes

In the description of statements and functions that follow, the following conventions are used:

1. Capital letters denote keywords, etc., which must be typed by the user exactly as shown (e.g. "PRINT", "RUN").
2. Lower case letters denote class(es) of items which may be used. The various classes are shown below. (e.g., "avar", "sexp")
3. Items enclosed in square brackets (e.g., " [, var] ") are optional.
4. Items enclosed in square brackets together with ellipses imply that the item shown may repeated any number of times. (thus " [, exp...] " is equivant to " [, exp, exp, exp...] ", etc.).
5. Multiple items in braces indicate that any one may be used. (e.g. $\left\{\begin{matrix}\text{END}\\\text{STOP}\end{matrix}\right\}$ END and STOP are equivalent statements).
6. Statement abbreviations - Basic will compare abbreviated statements (1 - N characters followed by period) and use first match. The statement will be LISTed in its expanded format. (Ex POS = POSITION). ? will be interpreted as PRINT

Types of items used herein

avar — Arithmetic VARiable, a storage location for a numeric value. Variable names are 1 to 120 alphanumeric characters and must start with an alphabetic character.
(e.g., "TOTAL", " COUNTER3")

svar — String VARiable, a storage location for a string of characters (bytes). Same name rules as "avar" except last character must be a dollar sign ("$") which is included in the count of characters in the name (e.g. "NAMES", "ADDRESS$")

mvar — Matrix VARiable, an element of an array (matrix) of numeric values. The name of the matrix variable is similar to a string variable name except that the last character must be a left parenthesis (e.g., "COUNTS (", usually seen in context as "COUNTS (CNTNUM)" indicating the CNTNUMth element of the array COUNTS.

var — VARiable, any of "avar", "svar" and "mvar"

aexp — an Arithmetic EXPression, generally composed of "aexp aop aexp " recursively where each aexp element may be an "lexp" "avar", "mvar", numeric literal, or arithmetic function.

Examples: "3 + A", "2*SIN(30)", "2 + (1-A)"

lexp — Logical Expression, generally composed of "aexp lop aexp" or "sexp lop sexp"; a logical expression evaluates to "true" (represented numerically by a constant 1) or "false" (numerically, 0).

Examples: 1<2 is true, "CAT" = "DOG" is false

sexp — String EXPression, can consist of a string variable, string literal, or a function which returns a string value, No operators are allowed in a sexp.

Examples: ADDRESS$, "RIDGEMS, BROWN", CHR$(RND(0)*128)

aop — The arithmetic operators
+ - * / ^

lop — The logical operators
< <= = > >= <> =
AND OR NOT (special case; NOT is a unary op)

lineno	A literal number specifying a line number
iexp	An arithmetic expression (aexp) which is rounded to nearest positive integer value.
exp	Either aexp or sexp
adata	ASCII with or without quotes. Leading blanks ignored, quotes throws out.

Operator Precedence

The following order of operator precedence will be used (operators on the same line have equal precedence):

< <= > => = <>	when used for string comparisons
-	unary minus
^	a number raised to a power
* /	
+ -	when used as binary operators
< <= = > => <>	when used for arithmetic comparisons
NOT	
AND	
OR	

Precedence may always be overridden by parenthesis, but no binary operator may operate on mixed strings and numeric elements (i.e., "A$ = C$ + B" is legal but "A$ = (C$ + B)" is not).

AND	-	a logical operator which requires both the left and right arguments to be true for the statement to be true.
NOT	-	a logical operator which reverses the truth of the argument following it.
OR	-	a logical operator which requires either the left or right argument to be true for the statement to be true.

Fundamental Control Statements

GOTO lineno - transfer execution to line "lineno"
GOSUB lineno - call a subroutine which begins at "lineno"
RETURN - return from a subroutine to next statement after last executed GOSUB

ON iexp {GOTO / GOSUB} lineno [, lineno.....]
GOTO or GOSUB the 1st, 2nd, 3rd...lineno in the list if iexp evaluates to 1,2,3...respectively. (if iexp =0 or is >than the number of lineno's in the list it falls thru to next line).

RUN begins execution of program currently in memory. All variable values are set to zero, all strings and arrays are undimensioned.

{END / STOP} terminate program execution, END may be omitted if last line of program

NEW erase program currently in memory

IF aexp THEN {lineno / statement...}

If the aexp evaluates to zero, control passes to the next sequential line. If non-zero, the statement following the THEN (and any subsequent statements in the line) is executed. The form "THEN lineno" is equivant to "THEN GOTO lineno"

FOR avar = $aexp_1$ TO $aexp_2$ [STEP $aexp_3$]
NEXT avar

When FOR is executed, avar is given the value $aexp_1$
When NEXT is executed, $aexp_3$ (which takes the value +1 if not given) is added to avar. If avar is not then greater than $aexp_2$, control passes to the statement following the FOR: all loops executed at least once.

$aexp_3$ may be positive or negative.

Avar is required for NEXT statement.

CLR — clears arrays

POP — (see page 10)

TRAP aexp — on error goes to expression aexp

aexp > 32767 & < 65535
clears trap

MISCELLANEOUS STATEMENTS

```
READ    var [, var...]
DATA    adata [, adata...]
RESTORE [lineno]
```

DATA statements allow information to be stored in a program without the need for any file I/O to retrieve it. The RESTORE statement is similar to a random file position capability, and READ is similar to file input. Statements may be located anywhere in program. Comma in data statement are defined as carriage return regardless of quote.

```
DIM {svar(aexp)                    }[{,svar(aexp)                    }...]
    {Mvar (aexp [, aexp])}         {,Mvar(aexp [, aexp])}
```

Allows the user to declare the "SIZE" of an array (number of elements) or string (number of characters) variable. Any attempt to access an element/character outside the declared size will result in an error.

CONT — CONtinue allows the user to continue program execution after a STOP END or BREAK at next line number not next statement.

REM — REMark is a "dummy" statement which allows the programmer to include comments for clarification.

{DEG / RAD} — Affect only the trig functions (SIN,COS, ATN). Select DEGrees or RADians as the argument/result values. Basic defaults to radians

BYE — Exit from BASIC back to the resident OS and/or console processor.

```
[LET] svar   = aexp
or [LET] {mvar} = aexp
         {avar}
```

Assigns the value of the expression on right side of the equals sign to the variable element specified on the left side.

INPUT/OUTPUT STATEMENTS

In all I/O statements the "#aexp" is an optional file specifier in the range 1-7. Omitting aexp causes I/O to use the keyboard.

PRINT [#aexp{,;}] exp [{,;} exp....] {;}

Prints the ASCII equivalents of the given expressions to the file specified. sexp's are simply output (without any conversion — i.e., the full 8-bit byte is output) from their beginning to their length. aexp's are converted to printable form. A comma following an exp causes "tabbing" to the next tabular column. A following semicolon causes no spacing.

LPRINT - Prints to printer, requires no OPEN, CLOSE or file specifier

INPUT [#aexp,] var [, var.....]

Requests ASCII input from the specified filenumber. If "var" is actually "svar", accepts a string of characters without transformation until an end-of-line is detected (end-of-line is carriage return (0D)). For an "avar" "nvar", numeric data is converted from ASCII to internal form. Numeric data may be delimited by a comma or an end-of-line.

LIST "flspc" [lineno [, lineno]]
LIST [lineno [, lineno]]

Lists program currently in memory to the specified file or device. If two lineno's are given, only the lines from the first through second lineno's (inclusive) are listed. If a single lineno is given, only that line is listed.

ENTER "flspc" brings program saved by LIST (ASCII source) back into memory

LOAD "flspc"
SAVE "flspc"

Provide a means of retaining the program currently in memory on a mass storage device (e.g., cassette) and then later recalling the program from the device. LOAD & SAVE process program in token format.

CLOAD LOAD and SAVE to cassette.
CSAVE

OPEN #aexp, $aexp_1$, $aexp_2$, "flspc" Opens file for input or output and assigns it to file specifier
#aexp = file specifier (0 - 7)
$aexp_1$ = open mode(4 = input 8 = output)(AUX1)
$aexp_2$ = device dependent (AUX2)

CLOSE #aexp

XIO CMD, #aexp, $aexp_1$, $aexp_2$, "$flspc_1$, $flspc_2$"

CMD

32 RENAME ($flspc_1$ = old name, $flspc_2$ = new name)

33 DELETE FILE

34 FORMAT DISK

35 LOCK

36 UNLOCK

37 POINT

38 NOTE

flspc = Dev [: file name[. file ext]]

Dev = D[1 - 8] for disk

P[1 - 8] for printer

C[1 - 8] for cassette

file name = 1 - 8 alphanumeric characters
the first character must be alpha

file ext = 0 - 3 alphanumeric characters

file name and extension are only used for disc files.

$aexp_1$ = 4 for input (AUX1)

8 for output

$aexp_2$ = device dependent information (AUX2)

GET #aexp; avar Inputs a single byte
from file specified by #aexp and stores in VAR

PUT #aexp; aexp Outputs single
byte to file specified by #aexp.

ex

```
GET #1: A
A $ = CHR $ (A)
PUT # 1; ASC ("A")
```

STATUS #aexp; avar-stores status of device into variable.

INPUT/OUTPUT STATEMENTS (continued)

RUN ["filspec"]

A variation on the normal RUN statement, this statement actually simply performs two statements, in order:

```
LOAD "filspec"
RUN
```

Note RUN works only with SAVE files

Functions return a value which is a transformation of some kind on the given argument.

Arithmetic Functions

LOG	(aexp)	log base 10
INT	(aexp)	integer, returns next smaller integer
ABS	(aexp)	absolute value
SGN	(aexp)	sign of argument return +1, 0. or -1
SQR	(aexp)	square root
LN	(aexp)	natural log
EXP	(aexp)	exponential (e^x, e=2.7182818...)
FRE	(dummy)	returns number of bytes of memory still available
RND	(aexp)	returns a pseudo-random number in the range 0. to 1. but never returns 1. Basic uses hardware random number generator so numbers are not repeatable. Aexp is a dummy argument.
* SIN	(aexp)	trigonometric sine
* COS	(aexp)	trigonometric cosine
* ATN	(aexp)	trigonometric arctangent

*(DEG & RAD affect mode of calculation of trig functions)

STRING FUNCTIONS/STRING-RELATED FUNCTIONS

LEN	(sexp)	returns the length (in bytes) of the argument string
VAL	(sexp)	returns the equivalent numeric value of arg.
STR$	(aexp)	returns a string looking like the PRINTed form of the arg.
CHR$	(aexp)	transforms a numeric value (0-255) to a string byte
ASC	(sexp)	transforms a single string byte to a numeric value

MACHINE ACCESS FUNCTIONS, ETC.

PEEK (iexp)
POKE $iexp_1$,$iexp_2$ — returns contents of byte in memory at address iexp; changes contents of byte at memory location $iexp_1$ to the value of $iexp_2$. (iexp 1 = 0-65535, iexp 2 = 0-255)

USR ($iexp_1$[,iexp...]) — calls machine language subroutine at address $iexp_1$, passes other iexp's to routine as arguments. Allows for return of a single numeric value (iexp). Arguments are stored in stack and occupy 2 bytes each. The last byte in the stack is the number of arguments. The two byte value returned by the machine language function must be stored in.

POP — Causes last item in users GOSUB/FOR stack to be pulled off and discard

ADR ( ) — returns address of string or numeric array :

NOTE	#aexp, var_1, var_2,	returns current sector # and byte within sector for specified disk file.
POINT	#aexp, var_1, var_2,	sets current sector # and byte within sector for specified disk file.

1. There can be only one STR$ and only one CHR$ in a logical compare. i.e. A = STR$ (1) > STR$ (2) is not valid

2. Val will look at 1st 255 characters only.

SPECIAL MACHINE-DEPENDENT STATEMENTS & FUNCTIONS

Graphics & Screen Control

GRAPHICS aexp -selects one of 12 (0-11) screen modes. Modes 1-8 are screen. (See page 16)

COLOR aexp -selects color register for PLOT and DRAWTO

PLOT aexp, aexp -PLOT a single point at X, Y

POSITION aexp 1, aexp 2 -Positions cursor at X, Y

SETCOLOR $aexp_1$, $aexp_2$, $aexp_3$ - set color register (0-4) to color/lum.
$aexp_1$ = color register
$aexp_2$ = color (0-15)
$aexp_3$ = luminescence (0-14, even numbers only)

DRAWTO $aexp_1$ $aexp_2$ -draw line from previous PLOT to X, Y $aexp_1$ = X $aexp_2$ = Y

LOCATE $aexp_1$, $aexp_2$, var = Store color value from screen position aexp 1, aexp 2 into VAR

SOUND $aexp_1$, $aexp_2$, $aexp_3$, $aexp_4$ - set sound registers to frequency, control and volume
aexp 1 = sound register (0-3)
aexp 2 = audio frequency 0 -255 (0= high, 255= low)
aexp 3 = audio control 0 -15 (10 = pure tone)
aexp 4 = volume 0 - 15 (0 = off 15 = load)

PADDLE (aexp) -returns the setting (0-255) of the paddle given by aexp (0-3)

PTRIG (aexp) -returns the status (0=OFF,1=ON) of the paddle trigger selected by aexp

STICK (aexp) -returns the setting (0-15) of the joystick designated by aexp(0-3)

STRIG (aexp) -returns the setting (0 or 1) of the joystick trigger designated by aexp aexp= joystick number (0-3)

DOS -transfers control to Disc Operating System (DOS)

ERROR REPORTING

Errors will cause a message of the form:

"ERROR xx at LINE nnnn".

The error number ("xx" above) may be in the range 1 to 199.

ERROR MESSAGES

ERRNUM 1 ; NO LINE # FOR EXP IN ON GOTO or GOSUB STATEMENT

ERRNUM 2 ; MEMORY FULL

ERRNUM 3 ; VALUE ERROR

ERRNUM 4 ; VARIABLE TABLE FULL (TOO MANY VARIABLES)

ERRNUM 5 ; STRING LENGTH ERROR

ERRNUM 6 ; READ OUT OF DATA

ERRNUM 7 ; VALUE NOT + INTEGER

ERRNUM 8 ; INPUT STMT ERROR

ERRNUM 9 ; ARRAY/STRING DIM ERROR

ERRNUM 10 ; ARG STACK OVERFLOW

ERRNUM 11 ; FLOATING POINT OVERFLOW

ERRNUM 12 ; LINE NOT FOUND (GOSUB/GOTO)

ERRNUM 13 ; NO MATCHING FOR

ERRNUM 14 ; LINE TOO LONG

ERRNUM 15 ; GOSUB/FOR LINE DELETED

ERRNUM 16 ; BAD RETURNS

ERRNUM 17 ; EXECUTION OF GARBAGE

ERRNUM 18 ; STRING DOES NOT START WITH VALID NUMBER

ERRNUM 19 ; LOAD PGM TOO BIG

ERRNUM 20 ; Device # >7

ERRNUM 21 ; Not a LOAD file

DECIMAL	HEX		BASIC I/O ERROR CODES
128	8Ø	-	BREAK KEY ABORT
130	82	-	NON-EXISTENT DEVICE
131	83	-	DATA ERROR
132	84	-	INVALID COMMAND
133	85	-	DEVICE OR FILE NOT OPEN
134	86	-	INVALID IOCB NUMBER (not stored in IOCB!)
136	88	-	END-OF-FILE
138	8A	-	DEVICE TIMEOUT (DEVICE DOESN'T RESPOND)
139	8B	-	DEVICE NAK
140	8C	-	SERIAL BUS INPUT FRAMING ERROR
141	8D	-	CURSOR OUT OF RANGE
142	8E	-	SERIAL BUS DATA FRAME OVERRUN
143	8F	-	SERIAL BUS DATA FRAME CHECKSUM ERROR
144	90	-	DEVICE DONE ERROR (RESPONDS WITH INVALID DONE BYTE)
145	91	-	READ AFTER WRITE COMPARE ERROR (DISK HANDLER)
160	AØ	-	DRIVE # ERROR
161	A1	-	TOO MANY OPEN FILES (NO SECTOR BUFFER AVAILABLE)
162	A2	-	MEDIUM FULL (NO FREE SECTORS)
163	A3	-	FATAL SYSTEM DATA I/O ERROR
164	A4	-	FILE # MISMATCH
165	A5	-	FILE NAME ERROR
166	A6	-	POINT DATA LENGTH ERROR
167	A7	-	FILE LOCKED
168	A8	-	COMMAND INVALID (SPECIAL OPERATION CODE)
169	A9	-	DIRECTORY FULL (64 FILES)
170	AA	-	FILE NOT FOUND
171	AB	-	POINT INVALID

Arithmetic Precision and Methodology

A special form BCD (decimal)floating point arithmetic will be used. Numbers may be in the range of ± 0.9999...E+98 to ± 1.0000...E-98 (and zero, of course).

BCD is urged for ease of user understanding. The particular BCD implementation chosen will be faster than traditional binary floating point for adds and subtracts, slower for multiplies and divides, and only slightly slower for array element access.

Matrices

Are two-dimensional. Either dimension may have a value of 0 to 32767, except that total space is limited by memory size. Elements will be numbered from 0, thus DIM ARRAY (33) will allocate 34 elements and DIM MAT (3,4) will allocate 20 elements (3+1 by 4+1).
No matrix will be automatically dimensioned.

Strings

String arrays are not supported. Strings are DIMensioned to contain a fixed maximum number of characters. The characters in a string are numbered from 1 to the DIMensioned maximum.

Substrings are specified as follows:

Specified as	Destination String	Source String
STRINGS $	the entire string, 1 thru DIM thereof	from the 1st thru LENgth chars
STRINGS $ (n)	from the nth thru DIMth character	from the nth thru LENgth chars
STRINGS $ (n,m)	from the nth thru mth character	same as dest.

It is an error if either the first or last specified character (n and m, above) is outside the DIMensioned size. It is an error if the last character position given (explicitly or implicitly) is less than the first character position. (e.g., PRINT A$ (20) is an error if LEN (A$) <20.)

Concatenation is not directly supported but may be performed as follows:

```
LET A$ = B$
LET A$ (LEN (A$) +1) = C$
```

(The above is equivalent to A$=B$+C$ as found in some BASICs).

Statements in General

Multiple statements may be entered for any given lineno by separating them with a colon.

The following statements must be the last on any given line. (This ensures program integrity in case of error or editing yet still allows CON at any time.)

END
STOP
REM

Syntaxing and Tokenizing

Statements are checked for syntax on entry. An error will cause the cursor to be repositioned at the point of detectable error (which is not necessarily the actual point of error: e.g., FOR I = ATOB STEP C is an error because spaces are required around the word TO; but the point of detectable error is the word STEP since ATOB is legal variable name).

Variables will be placed in a variable table at entry time (not when the program is RUN).

Each variable and/or reserved word in a program will occupy 1 byte.
String constants will occupy string length +1
Numeric constants will occupy 7 bytes.

Deferred and Direct Modes

All statements noted herein may be used with or without a line number (herein after called "deferred" and "direct" statements). Some statments may have little meaning in one or the other mode; others may have unexpected meanings. - Some of the less obvious are noted below:

In Deferred Mode

RUN	is equivalent to encountering END and then the operator typing RUN.
CONT	is a no-op. That is, it has the same effect as a REmark.
NEW	are equivalent to encountering END and the operator typing NEW.

In Direct Mode

DATA
REM
END
STOP

have no meaning and cause no action at all.

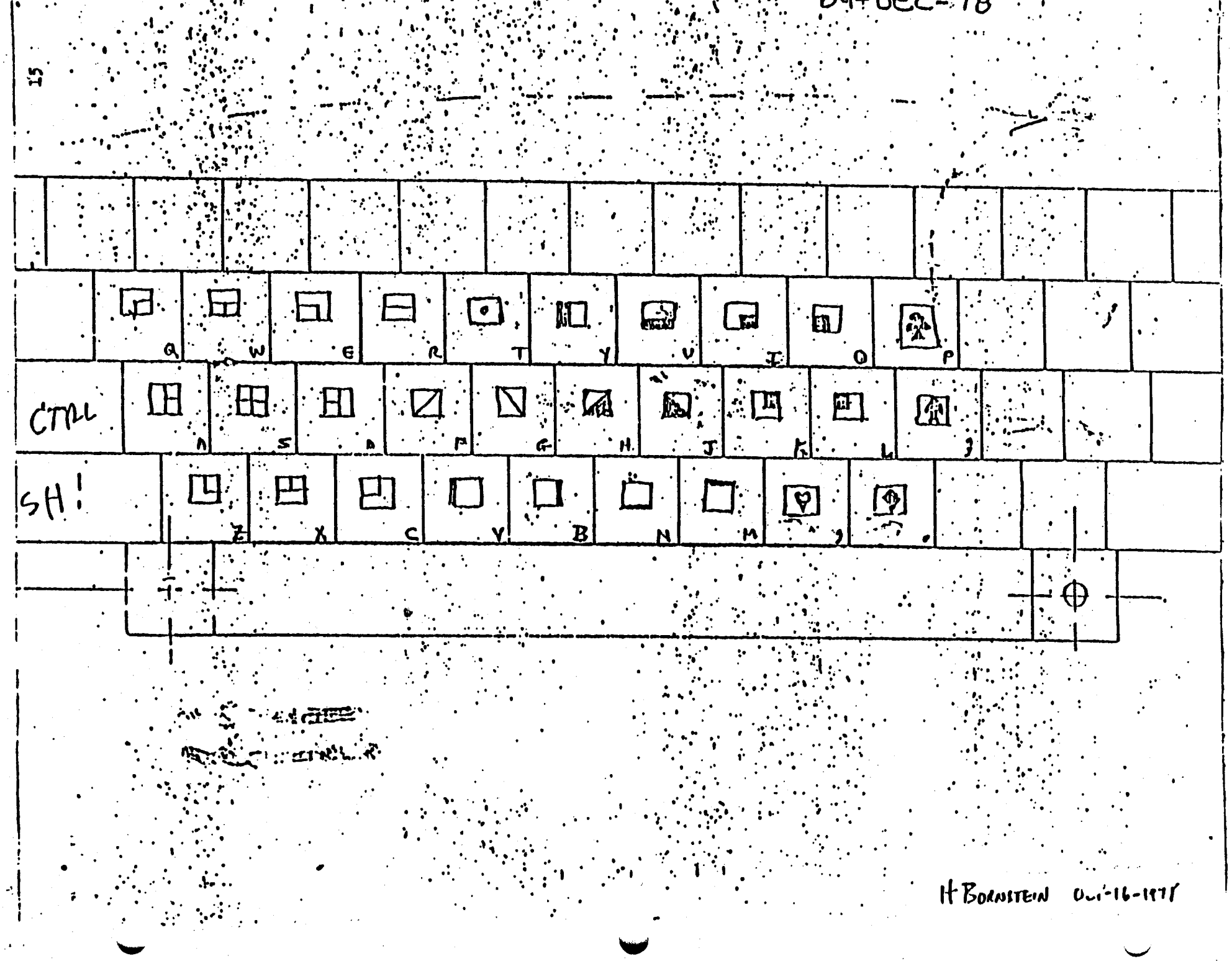
GOTO — starts (restarts) program execution at the lineno noted in the GOTO statements.

GOSUB — calls the subroutine at the given lineno; Upon RETURN control returns to direct mode statements.

Graphics Control Characters

Grahics control characters may be inserted into Basic PRINT strings by preceding each character by an escape character. When the program is LISTed the control character is displayed as a special graphic. When the program is RUN the control character is executed.

Control Character	Special Display character
ESC	Ec
↑	↑
↓	↓
←	←
→	→
CLEAR	↰
BACKSP	◀
TAB	▶
	Inverted Chr
DEL LINE	↑
INS LINE	↓
CLR TAB	←
SET TAB	→
BELL (cntrl 2)	↰
DEL CHR	◀
INS CHR	▶



H BORNSTEIN Oct-16-1977

SCREEN MODES AVAILABLE THROUGH BASIC

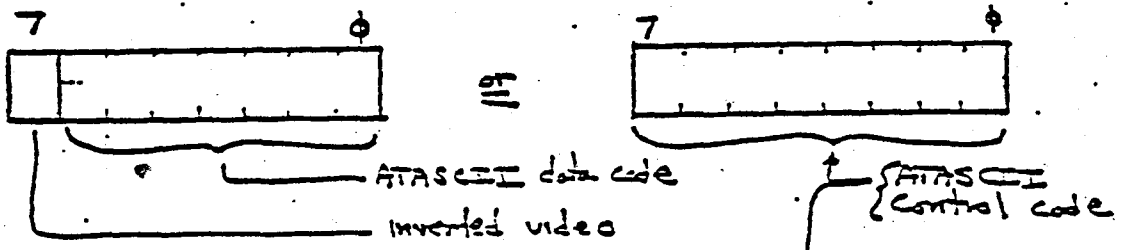
MODE	TYPE	# OF SCREEN POSITIONS			OBJECTS AVAILABLE ..... BACKGRND =OBJ 0 *	DEFAULT COLORS/ OBJECTS	MEMORY REQR- MENTS BYTES	CHARACTER SETS AVAILABLE
		HORI- ZONTAL	VERTICAL NORMAL	VERTICAL WITH TEXT WNDW				
0	TEXT	40	24	N/A	OBJECT2 LUM** OBJECT3	WHT ON GRN-BLU OBJ2 LUM**	993	ALL
1	TEXT	20	24	20	OBJECTS 0-4	WHT ON GRN-BLU	513	UPPER CASE & NUMBERS OR LOWER CASE & GRAPHICS CHARACTERS
2	TEXT	20	12	10	OBJECTS 0-4	OBJ 1	261	
3	GR	40	24	20	OBJECTS 0-3	LIGHT BLUE ON BLUE OBJ1	273	NONE
4	GR	80	48	40	OBJECTS 0-1		537	
5	GR	80	48	40	OBJECTS 0-3		1017	
6	GR	160	96	80	OBJECTS 0-1		2025	
7	GR	160	96	80	OBJECTS 0-3	LIGHT BLUE ON BLUE OBJ1	3945	
8	GR	320	192	160	OBJECT2 LUM** OBJECT3	LT BLUE ON BLUE OBJECT2 LUM**	7900	

* EACH OBJECT (EXCEPT AS NOTED) MAY HAVE ITS COLOR SET TO ONE OF 16 DIFFERENT COLORS. SEE "SETCOLOR" FOR COLORS AVAILABLE.

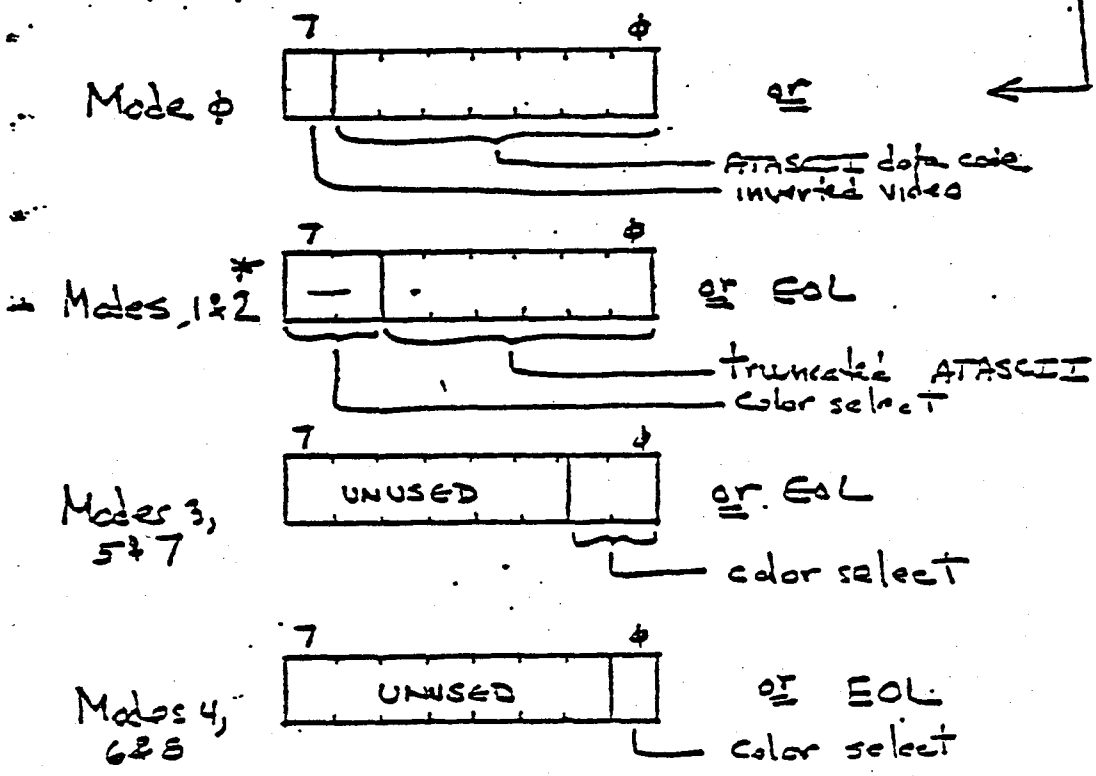
** LUMINANCE (BRIGHTNESS) ONLY CAN BE SET FOR THESE OBJECTS. LUM CAN BE 2 (LIGHTEST SHADE), 4, 6, 8, 10, 12, OR 14 (DARKEST SHADE).

Keyboard / Display Data Formats

1. Keyboard data (mode independent)



2. Display data



* Note: A "E" character + color select = 10 is indistinguishable from an ATASCII EOL.

	CLOSE ALL BUT IOCBØ			PRINTS READY
IMMEDIATE	Y			Y
RUN	N			N
STOP	N			N
END	Y			Y
⟨BREAK⟩ while running	N			N
⟨BREAK⟩ while stopped	N			N
CONT	N			N
BYE	Y			N
DOS	Y			N
⟨OFF END⟩ *	Y			Y
⟨ERROR⟩ I/O	except one in error N			N
⟨ERROR⟩ I/O	N			N
NEW	Y			Y
ENTRY	N			Y

*prog. commands

X = whatever's easy, doesn't matter.

BASIC ABBREVIATIONS

B.	BYE
:LOA.	CLOAD
:CL.	CLOSE
C.	COLOR
CON.	CONT
CS.	CSAVE
E.	ENTER
D.	DATA
DE.	DEG
DI.	DIM
DO.	DOS
DR.	DRAWTO
F.	FOR
GE.	GET
GOS.	GOSUB
G.	GOTO
GR.	GRAPHICS
I.	INPUT
LE.	LET
L.	LIST
LO.	LOAD
LOC.	LOCATE
LP.	LPRINT
N.	NEXT
NO.	NOTE
O.	OPEN
PL.	PLOT
P.	POINT
POK.	POKE
POS.	POSITION
PR.	PRINT
?	PRINT
PU.	PUT
REA.	READ
R.	REM
RES.	RESTORE
RET.	RETURN
RU.	RUN
S.	SAVE
SE.	SETCOLOR
SO.	SOUND
ST.	STATUS
STO.	STOP
T.	TRAP
X.	XIO