KBASIC Interpreter
BASKOM Compiler
$GRAF Graphic Driver

for

KONTRON PSI-80/900

H. Krieger
Mathenesserlaan 368a
3028 HB Rotterdam
Tel. (010)-771914

The KBASIC Interpreter and BASKOM Compiler/Interpreter are written for Kontron PSI-80/900 computers operating under KOS 4/5 and 6. KBASIC is the name of a greatly revised Psi-BASIC program. It is compatible with Psi-BASIC and occupies the same 18K of memory. $GRAF is a new graphic driver for use with KBASIC.

There have been a number of changes made in Psi-BASIC that involve corrections of errors, new commands and instructions, a revised input editor, and a shortening of the execution time of Basic programs. Larger programs, especially, will run in about 1/3 the time as formerly.

The input editor allows single stroke entry of Basic commands. It also checks and, in most cases, corrects syntax errors. While in the TRACE mode, the program line being executed is displayed along with the values being assigned to variables and loop indexes. In general, it is much easier to write and check out new programs.

Data types include hexadecimal constants, as well as single and double precision real variables. Program structures can be improved by use of WHILE/WEND and REPEAT/ UNTIL instructions. The BLIST command gives a formatted listing of the program structure.

Subroutines and functions can be identified by labels. The Labels are used in place of line numbers in GOTO, GOSUB, and IF/ THEN/ELSE statements, as well as for function names in any expression. The arguments may be passed to and from subroutines or functions. New MERGE and CHAIN instructions permit large programs to be broken into smaller modules. READ and WRITE Binary instructions allow efficient Input-Output and storage of numeric data files. The GET instruction allows better integration with I-O drivers for real-time applications.

BASKOM is a Compiler/Interpreter used in combination with KBASIC. It converts all jump and variable references in .BAS type programs to specific memory addresses. It produces a .BAC type file which executes much faster than the normal Basic program.

$GRAF is a revised $GRAP graphic driver for use with KBASIC under KOS 4/5/6. It will generate vectors with a variety of line types, fill rectangles with a variety of lines and patterns; and allow a greater choice of text formats and angles. $GRAF includes arc and circle generation; as well as a WINDOW command that divides the CRT into four quadrants. The new graphic instructions in KBASIC make it easy to produce bar graphs and pie charts. $GRAF is about 30% faster than $GRAP, and uses the same amount of memory.

Contents

Summary of Additions to Psi-BASIC

Keyboard and display

Lower Case keys:	Basic Word input with a single stroke
Cursor Up/Down keys:	Scrolling of program listing
CNTR-T Key:	TRACE enable/disable during program run
Tracing mode:	Program lines, variables, and indexes shown
Error reports:	"_" indicates position of errors in line

Commands, Instructions

BLIST:	Structured PLIST of Basic Program
INFO:	Display .INF type file as in: INFO HELP
LI Variable:	List lines containing Variable
LI #Number:	List lines containing Line Number
RL:	Relist last LIsted block of lines
VLIST:	List defined program variables
VKEY:	Define input string for "v" key
CHAIN:	Read in continuation of program
DAY$	Function that gives day and date
DEGREE / RADIAN:	Use deg./rad. for trig. functions
ELSE:	I-O error condition test
FIND #N:String	Move file to next line having String
FRAC(Variable):	Give fractional part of real
FOR/NEXT:	Integer variable used as index
GET #N:	GET 8 or 16-bit input from I-O channel
GOSUB..(parameters):	Pass parameters to subroutines
KOS $K:	KOS command as string variable
MERGE:	Merge ASCII code program
OPEN #C:, etc.:	Channel number given by variable
PLOT X,Y,P (,R,S):	Additional graphic instructions
POKE Adr, D1,..,Dn:	Poke sequence of values into memory
POS(String, Substring):	Finds position of Substring in String
REC(0)	Give current file record number
REPEAT / UNTIL cond.:	Repeat loop until cond. is true
RETURN(parameters):	Pass parameters back to calling statement
SINGLE / DOUBLE:	Single / Double (normal) precision
STEP:	Single step in TRACE mode
STR$(X, Format):	Formatted output of STR$(X)
SWAP:	Exchange variable values
TIMER(arg):	Read (or read and reset) 20 msec. counter
TONE M,N:	Acoustic tone 'M' for duration of 'N'
USING format:	"(", "&", and ";" format characters
VAL(String):	Leading letters and spaces ignored
VPTR(Variable):	Give address pointer of variable
WINDOW N:	Selection of graphic area on CRT
WHILE cond. / WEND:	Execute loop while cond. is true
WRITE #N: / READ #N:	Binary Input-Output of data
'label(arguments)	User defined function or subroutine

Program additions

Labels:	Label names for line numbers or functions
Data types:	Hexadecimal; Single precision; Repeat char.

Program Production

Single key input can be used to enter 26 instructions and commands. The lower case letters are assigned as follows:

LEN(	ASC(	VAL(	LEFT$(	MID$(	RIGHT$(	STR$(	CHR$(	INT(	ABS(
q	w	e	r	t	y/z	u	i	o	p

FOR	STEP	NEXT	THEN	ELSE	REM	GOTO	GOSUB	RETURN
a	s	d	f	g	h	j	k	l

PRINT	USING	INPUT	VKEY	SAVE	LOAD	RUN
z/y	x	c	v	b	n	m

The lower case letters are used in the normal manner after a 'REM' (entered by use of 'h') or after a quotation mark. Inorder to enter a lower case letter when editing a line, you may find it necessary to first enter a quotation mark and then delete it.

The 'v' key is programmable. It can be defined at any time to give an expression containing up to 40 characters.

Use of 'b' for SAVE will also display the name of the last file that was either loaded or saved.

The input editor checks for syntax errors, and also in many cases, will correct the error. If a syntax error occurs in the line being entered, the line will reappear in the edit or FEtch mode with an underline character at the place where the error was found. Missing parentheses will be added or incorrect characters changed. If you are satisfied with the correction, you can press RETURN and have the line accepted. Otherwise, the line may be corrected and entered in the normal manner. It is not necessary to delete the underline mark unless it occurs within quotation marks. If a line with syntax errors is not corrected, the line will continue to reappear until you either correct it or rub it out.

The line appearing in FEtch mode may have a length greater than that defined by LINELEN, or the default length (80). The extra characters may be left in the program line or deleted, but cannot be replaced.

The CURSOR UP key will enable the upward scrolling of program lines. The scrolling begins with the line after the last one displayed thru use of LIst or FEtch. Scrolling is done in the FEtch mode if the last command was a FEtch, otherwise it is done in the LIst mode.

When the cursor is at the left side of the CRT, depressing CURSOR DOWN activates the KOS P function. This allows normal CRT paging. When the cursor is not at the side of the screen, then CURSOR DOWN acts as a Single Character RUBOUT, erasing the character located to the left of the cursor.

Commands

The LIST command has been expanded. It is possible to list all lines containing a particular variable or jump address. LIST followed by a variable name will list all the lines in which the variable appears. LIST# followed by a line number will list all lines in which there is a GOTO, GOSUB, THEN, or ELSE with the given line number.

LIST is also used to modify the names of variables. A variable name coming before the slash is replaced by the letters or numbers coming after the slash. The number of characters in a name does not change. The names of Reals, Integers, and Strings remains unique, however, no distinction is made between variables with or without arrays.

Listings will show a "'" after the line number whenever there is a reference in the program to the line number in GOTO type statements. REM is replaced by the mark "!" to make the listing more readable.

```
LI 100-200        List lines 100 to 200
LI A1             List lines containing variable A1
LI AR(            List lines containing the array AR(..)
LI A*             List lines with variables starting with A
LI #1234          List all lines referring to line 1234
LI LABEL          List lines containing label 'LABEL
RL                Relist lines 100 to 200
LI X1/AB          Change variables X1 or X1() into AB or AB()
LI NAME$/D        Change NAME$ to DAME$
```

-BLIST- is similar to PLIST, however, it gives a more structured format with each instruction printed on a separate line. An example of the BLIST output is shown in the description of Labels.

-RL- stands for ReList. This command will repeat the last LIST command and the given starting and ending line number. This allows continual viewing of the block in which lines are being edited.

-VLIST- lists all variables defined in the program. Variables are given with the first two or three characters in the variable name (the third character may be a prime or tic mark) together with the integer (%), or string symbol ($), and parentheses for an array.

-DELETE- replaces 'DE'. This is to avoid accidental deletions.

-INFO- displays 20 lines from ASCII file (.INF is default type). The Space bar then brings in the next 20 lines. 'INFO HELP' gives a summary of all KBASIC commands, instructions, and error messages.

-RNB- Non-existent line numbers are blanked out in GOTO/GOSUB/THEN/ELSE expressions inorder to avoid wrong jumps to a renumbered line.

-VKEY "String"- assigns the String to the lower case V key for use when keying in program lines. A String can hold up to 40 characters.

Instructions

-"C"(N)- is a string function that repeats a character 'C' N times. A$=" "(72)+CHR$(13) gives a string with 72 spaces followed by a CR.

-CHAIN "File"- is similar to LOAD. It is used the same way as MERGE except that the file is a '.BAS' type. All variable are saved and computation continues at the first line of the chained program. The chained program file cannot be longer than the main program section.

-DAY$- is a function that gives the day along with the system date. The date is set with the KOS system program 'DATE'. The current Day and Date is displayed when KBASIC is loaded into memory.

-DEGREE- lets use of degrees with SIN, COS, TAN, and ATN functions.

-DOUBLE- 13 digit accuracy is restored for all operations. This is the default condition which is set when a RUN command is given.

-FRAC(Variable)- returns the fractional part of the variable. This is equivalent to: FRAC(X) = X - INT(X).

-FOR / NEXT- loops can be used with integer variables as the index. In this case, the starting, end, and step values must also be given as integers. The range for the index is 0 to 32767. The step value, however, can be a positive or negative integer.

-GOSUB Line no. (Argument1, Argument2, ...)- GOSUB statements can include arguments whose values are passed to the subroutine referred to by the GOSUB statement. These arguments can be in the form of expressions or variables. GOSUB arguments are assigned to variables in a subroutine by statements of the form: Var1=* : Var2=* : etc.

-RETURN (Argument1, Argument2, ...)- The RETURN statement is also used to return values to the calling statement. It is similar to a GOSUB with parameters, but works in the reverse manner.

```
100  GOSUB 200 (2*X(I)+C, 3*Y(I)+C, R)
110  X=*: Y=*: PRINT X,Y
:
:
200' P1=*: P2=*: P3=*                      P1, P2, P3 are used as
210  P1=P3*COS(P1): P2=P3*SIN(P2)          local variables in this
220  RETURN (P1, P2)                       example.
```

Statements of the form 'Var = *' are used to pick up the values of arguments passed from GOSUB adr() statements or 'label() functions. They are the the first statements found in a subroutine, and they must agree in sequence and number with the passed arguments.

-KOS- A KOS command must be given as a string or a string variable. 'KOS I' would be given as KOS "I" or KOS I$, where I$ = "I".

-MERGE "File"[,"DElete lines"]- is the same as ALOAD with an optional DELETE. The arguments may be strings or string variables. Any file name can be given, but the data must be the same as in .BSC type files. The given lines will first be deleted and then the file will overlay or merge with remaining program lines. Computation will continue at the first program line. All variables are saved. The continuing program cannot be longer than the starting program. The DELETE option is used to erase blocks containing lines that are not written over by new lines.

MERGE "PROG2","10-80", will delete lines 10 to 80 in the old program and read in PROG2.BSC. Computation continues at the first line.

-POS(String, Substring)- is a function which gives the position of a substring within a string. The arguments may be either strings or string variables. If the Substring is part of the String, then the POS function returns the first location of the beginning character of the Substring within the String. If the Substring is not found within the String, then the value of the function is zero.

-RADIAN- sets the default condition of radian units for arguments.

-REPEAT- marks the start of a 'Repeat Until Condition True' loop.

-SINGLE- Operations involving multiplication and division are done in single precision with 7 digit accuracy. Additions or subtractions are not affected. The use of SINGLE will shorten multiplication and and division times by 40%. Functions such as SQR, SIN, etc. as well as matrix operations are done faster with little loss of accuracy.

-STEP- allows stepping thru a program in TRACE mode. Pressing any key, except ESCAPE or CNTR-T, permits a single program statement to be executed with a display of the line containing the statement. Also displayed are the new values of variables and FOR/NEXT indexes. The ESCAPE key can be used to stop execution. The values of other variables may then be displayed by use of the PRINT command. STEP and TRACE modes are disabled by the CNTR-T key or use of 'TRACEOFF'.

-STR$(X, Format)- allows the formatting of the converted numerical value of X. The Format can be a string or string variable. Its form is the same as that in a PRINT USING statement. However, use cannot be made of the hexadecimal format for STR$. The unformatted STR$(X) will give X in exponential format.

-SWAP X,Y- exchanges the values of two variables of the same type. String variables must also have the same length.

-TIMER (N)- This function gives the count in a 20 msec. increment counter, that cycles continuously from 0 to 65535. The timer runs as a background task. It is initialized upon the loading of KBASIC, and then deactivated upon the return to KOS. If the list of active tasks is already full, then the timer is not activated and a warning is given. The timer runs for 65535*.02 seconds before returning to zero. If N=0, the count is given, and then reset to zero. TIMER(N) can be used to measure the intervals in real-time activities.

-TONE [M, N]- Sets the tone M, and gives the acoustic output for a duration of N * 5 ms. The range for M is 1 to 4095; N is 1 to 255. Freq. = 125000/M. For the note A in c-major(f=880), M=142 is used. If [M, N] is not given, then the last tone selected is used for an acoustic signal. 'TONE' is then equivalent to: PRINT #1: CHR$(7); The normal signal (M=127) is reset by RUN, or an error stop.

-UNTIL condition- Until the condition is true, the statements in a loop, starting at the REPEAT, are repeated. Entry to a loop must be at the REPEAT command. Exit from a loop is via the UNTIL statement.

-USING Format- The use of the character '(' as the first element in the format (before + or -) will suppress blanks before all numbers. This should be used when writing formatted numbers to a file. It would prevent having incorrect space delimiters in a file that may be read back with an INPUT statement.

The format "&" causes PRINT USING to output hexadecimal equivalents of real or integer variables. The use of ";" instead of "." places commas before every third digit to the left of the decimal point. An extra "#" must be given in the format for every expected comma. Else a "#" is inserted into the number where a comma might be expected.

-VPTR(Variable)- is a function that returns a pointer to the first byte in memory of the given variable. The pointer address is a number in the range 0 to 65535. Integer variables are stored as 16-bit two's complement numbers. A real variable is represented by an eight byte BCD number. P = VPTR(VAR$) will give the position of the first ASCII character in a string. The current string length is found in P-2 and P-1. The maximum length of the string, that can occupy the same location, is found in P-4 and P-3. The value of VPTR(..) can be also be gotten as a hexadecimal number 0 to FFFFH.

```
A%=100: PRINT PEEK(VPTR(A%))              gives an answer of:  &64
A$="ABC": PRINT CHR$(PEEK(VPTR(A$)))      gives an answer of:  A
```

-WHILE condition- As long as the condition is true, the statements in the loop, up to its corresponding WEND, are executed. A loop must be entered only via the WHILE statement.

-WEND- indicates the end or exit point of a WHILE/WEND loop.

Input - Output Instructions

The channel number 'n' may be given by either a constant or variable in all I-O instructions.

```
10 A=10
20 OPEN #A: "FILE1"
30 INPUT #A: L$
```

I-O drivers OPEN'ed on channels 1-9 may, or may not, have a '$' at the start of the driver name. "$SIOA" is the same as "SIOA". Driver names with less than four characters must be filled in with spaces, i.e. "$MON ". However, when an IEEE driver is activated, its address is given by a single character.

```
10 OPEN #5:CHR$(5)                    Opens address 5 in $IEEE
```

-GET #n: A$- is used to input 8-bit characters from a file or I-O driver. The character includes a parity bit. Files which contain non ASCII characters, or files without CR delimiters, can thus be entered into a program. The user must then make his own test for an EOF (0FFH). The 'GET #n:A$' differs from the 'GET A$' instruction in that 'GET #n:' always waits until a character is read in.

-GET #n: I%- with an integer variable, is used to read in a 16-bit number from an input driver or a file. In a driver (#9 or less) the the Low-order byte is returned thru the IY stack at (IY+7). The High order byte, however, must be loaded into (40H) for proper retrieval. From a file (#10-#14), two successive bytes are read in at a time.

GET #n: A$, I%, B$, J%, . . .- may use multiple variables.

-WRITE #n:- is used to write binary files of numeric or string data. Integer numbers use three bytes; normal precision decimal numbers use 8 bytes, and single precision numbers use 5 bytes of file space. The parameters must consist of single variable names. Arithmetical expressions cannot be used in a WRITE statement (as with PRINT).

-READ #n:- is used to read the binary data that has been written by a WRITE instruction. The sequence of data types must be the same in both the WRITE and READ instructions. Integer values in the file can be assigned to real variables in the READ instruction. However, decimal numbers in the file cannot be assigned to integer variables. Single or double precision numbers are interchangeable.

```
10 OPEN #10:"FILE"
20 WRITE #10: S$, X, Y, Z, I%, J%
.
.
490 OPEN #10:"FILE"
500 READ #10: T$, U, V, W, R, N%
```

-ELSE- can be used to test for an Input-Output error by placing it immediately after an I-O instruction. If no error occurs, then ELSE acts the same as a REM statement.

```
100 INPUT #10:A
110 ELSE PRINT "LINE 110",ERM(1): GOTO 130
120 PRINT A: GOTO 100
```

-REC (N)- is a function that gives the current record number in the file that was last used for an I-O instruction. The (N) is a dummy variable.

```
100 RECORD #10: REC(0)+1          Advances file to next record

100 RECORD #10: *
110 PRINT REC(0)                  Print last record number
```

A system crash may occur if the given RECORD number refers to a 48K segment that is not part of the file. That is, for a file of less than 384 records, do NOT use a number greater than 384. For a file of 385 to 768 records, do NOT use a number greater than 768 etc.

-FIND #n, A$- advances the file to the next line containing A$. If string A$ is not found, then an EOF error is given. The character '?' in A$ acts as a wild card, and replaces any character from the file. Upper/lower case letters are interchangeable. A program to display all lines containing the name Smith or Smyth is as follows:

```
10  OPEN #10:"NAMEFILE" : A$="SM?TH"
20  FIND #10:A$ : ELSE STOP
30  INPUT #10:B$ : PRINT B$ : GOTO 20
```

Increment and Decrement Operators

An Increment '++' or Decrement '--' Operator can be placed directly after a numeric variable in any expression. This has the effect of increasing or decreasing the value of the variable by 1 before the variable is actually used in the given expression. The following lines in a program give the same result:

```
10 A = 5 : A = A+1 : B = A
10 A = 5 : B = A++

100 I%=I%-1 : IF I%=10 THEN 300
100 IF I%-- =10 THEN 300
```

Integer variables have a value of -32768 to 32767. There is no overflow test on integer variables when using ++ or -- operators.

Labels

Labels are used at the beginning of statements inorder to identify statement locations in a program. Labels are placed directly after the line number. Labels names may be used in place of a line number in GOTO, GOSUB, or IF/THEN/ELSE instructions. A label must begin with an apostrophe, or tic mark, and consists of upper case letters and/or numbers. There is no limit on the number of labels in a program or the length of a label.

```
100  'ENTRY1: INPUT #C:Z: ELSE 'EOFSTOP
110' IF A(Z THEN GOSUB 'GETXY(Z) ELSE 'OUTPUT
120  X = *: Y = *: IF X(Y THEN 'OUTPUT: ELSE 110
130  'OUTPUT: PRINT X, Y, Z: GOTO 'ENTRY1
```

The tic mark after line number 110 indicates that there is a jump to the line from somewhere in the program. If the label before an ELSE has no parameters, then it is separated from the label with an ':'.

A label also identifies a user function. It is used in a statement the same way as any numeric function. However, a label function can not have label functions as arguments. It is an implied GOSUB, and the function is formed in the same way as a subroutine ending with a RETURN (var/expression) instruction. Multiple arguments may be used in the RETURN, but the function uses only the first one as a result. In the following example, X is calculated and then printed with the result rounded to N significant digits.

```
100  X = A+'MOD(B+C, D)                          Modulus function
110  PRINT 'ROUND(X, N)                          Rounding function
:
:
500  'ROUND: P1 = *: P2 = *
510  P3 = SGN(P1)*INT(ABS(P1)*10^(INT(P2))+.5)/10^(INT(P2))
520  RETURN (P3)
530  'MOD: P1 = *: P2 = *: RETURN (P1-P2*'FIX(P1/P2))
540  'FIX: PX = *: RETURN(SGN(PX)*INT(ABS(PX)))
```

Variable Names

Variable names may have Basic command words embedded in them. However, the name of a variable may not start with a Basic command. In addition to the first two two letters, or letter and number, a variable may be uniquely identified by an accent mark placed at the end of the the name. The second character in a name may also be an accent mark. To help give meaningful names, you may use decimal points in a variable name in any position after the first letter.

```
X, X', X'', X1%, X1'%    are all different variables.
XO, XO12, XO.STARTX       are names of the same variable.
```

Hexadecimal Constants

Hexadecimal constants may be used instead of decimal integers. They are identified by the character "&" in front of the number. They are given as a number: &0 to &FFFF. If more than four digits are given, only the last four are used. Values assigned to a real or integer variable can be in the range of 0 to 65535. Hexadecimal numbers may be written into the program or entered with the INPUT command. The 'PRINT USING "&"' command converts numbers in the range 0 to 65535 to the hexadecimal equivalents of &0 to &FFFF.

You would most likely use hexadecimal numbers in the PEEK and POKE instructions. The POKE form been modified so that a sequence of numbers can be entered following a given address. The parameters may be variables or constants. The address has a maximum value of &FFFF.

```
POKE  ADR, D1, D2, D3, . . . . . , DN
```

The output of a PRINT PEEK(ADR) command is a hexadecimal number. To obtain the decimal value of the byte read from a memory location, you could use the instructions: A=PEEK(ADR): PRINT A

```
POKE &40,&AB,100: PRINT PEEK(&40),PEEK(&41)   gives: &AB  &64
```

Program Testing

In TRACE mode, a program line is displayed as it is being executed. Each time a new value is assigned to a variable or loop index, it is displayed along with the variable name. String variables are shown with their length. In graphic mode, channel #2 is used for the TRACE output; the alphanumeric display may be thus directed to a printer.

The CNTRL-T key is used during program execution to toggle the TRACE mode on and off. You can thus check the progress of a program.

The STEP command sets the TRACE mode and allows the execution of a program statement by statement. It is stopped by the ESC or CNTRL-T key, or within a program, by the TRACEOFF instruction.

As only two characters (plus accent mark) are used for variables, there may be a mixup between different variable names that start with the same two characters. The 'VLIST' command is used to list the variable names; and 'LI Var1/Var2' used to modify one of them.

Differences with Psi-BASIC

The following errors in Psi-BASIC have been corrected:

- FEtch can now be used with line numbers (n * 256) - 1.
- IF ... THEN RETURN ELSE ... is now permitted.
- PRINT with a BASIC command as argument causes an error stop.
- SQR(X) and LOG(X) cause error stops if X is a negative number.
- STR$ can be chained such as in LEFT$(STR$(I), 3) etc.
- VAL ignores leading letters. 'VAL(DM- 123.--)' will gives -123
- FRE(0) includes space reserved by DIM for numeric variables.
  A minus value for FRE(0) means that more than 32K bytes are free.
- FOR/NEXT loops may be terminated prematurely with a GOTO outside
  of the loop without causing problems in subsequent loops.
- Array subscripts can also be arrays. 'A=A(1,B(2))' can be used.
- ON ERROR instructions are ignored for direct command errors.

Programs written with Psi-BASIC will run with KBASIC, however, an error message (address?) may occur due to a fault in the file SAVE command in Psi-BASIC. Inorder to correct the error, a program should be saved with ASAVE and then reloaded with an ALOAD command. KBASIC will ASAVE, ALOAD, and then SAVE the compacted program correctly.

REM coming after a statement must be preceded by the ":" separation.

RND(N) always gives a random number. It is no longer necessary to use RANDOM for seeding the RND(N) function.

Interrupt instructions are modified. At the start of an interrupt routine, defined by ON INTR GOSUB n, the interrupt is disabled. It is enabled again at the end of the routine by use of ON INTR RETURN. Interrupts may be disabled or enabled by ON INTR RES or ON INTR SET.

Error Messages

- If a CHAIN or MERGE file is too long, program variables are not saved, and the error message 105(media/memory occupied) is given.

- For DIM statements, containing a variable that had been declared in a previous DIM statement, a warning message will be given and computation continued. However, no message is given for repeated DIM statements occuring in a chained or merged program.

- When an error occurs which stops computation, the program line being handled at that moment will appear in FETCH mode, and an underline character '_' will be displayed at the operation being executed in the line when the error was found. Run time errors caused by floating point overflow, division by zero, or faulty square root and log function arguments will cause a stop with the appropiate error message. In case of underflow, the value is replaced with zero, and no error warning is given.

Most run time error messages have been shortened, and some of them combined. That is, no distinction is made between End of File or End of Data, or how a File or Medium is protected. The program line that is displayed following the error message, should indicate the nature nature of the error. However, you can use the PRINT ERM(0) command inorder to obtain the precise error code.

ERM	Error message	Likely reason for error
59	I/O unit?	File no. not in range 10-13
60	file not found	File inexistent or faulty diskette
61	end of file/data	End of record
62	illegal number	Value used is too large or too small
63	illegal number	RENUMBER parameter
65	line too long	Too many digits in INPUT value
66	stack overflow	More than 8 FOR/NEXT loops in use
67	file is open	Previous file is still open
68	repeated DIM	Array already dimensioned
71	loop?	Program ended in subroutine or loop
72	illegal number	Incorrect number used
73	address?	Jump address not found
74	FOR/NEXT index?	Mismatch in FOR/NEXT index variable
75	incorrect arguments	Inconsistent no. of arg. in GOSUB( )
77	insufficient memory	No. of variables too large
78	variable?	Variable not defined
79	math. overflow	Division by 0, or number) 1E+127
80	subscript) DIM	Subscript greater than DIM value
81	incorrect arguments	Illegal arg. in SQR or LOG function
82	loop?	No GOSUB, WHILE, REPEAT, or 'LABLE()
83	syntax	Incorrect operator or data
84	stack overflow	Too many GOSUB, REPEAT, or WHILE loops
85	variable mismatch	Mismatch in variable or data type
86	variable mismatch	String variable or data expected
87	end of file/data	No more data for INPUT or READ
88	integer overflow	Incorrect value used
89	line too long	ALOAD line longer than LINELEN value

KOS System errors

96	invalid parameter	KOS I-O error
98	I-O channel not ready	No diskette in unit
99	I/O channel not ready	IODC not invoked or channel not open
102	file protected	Diskette write protected
104	media occupied	Diskette is full
105	Media/memory occupied	Program too large, cannot be loaded
107	Data transfer error	Faulty or wrong diskette
109	File protected	File write/erase protected
110	Media/memory occupied	Faulty file block allocation

$GRAF

$GRAF is the name of a revised $GRAP graphic driver for use with KBASIC. It generates vectors with a variety of line types, fills rectangles with a variety of hatched lines and patterns, and allows a greater choice of text formats and angles. $GRAF includes arc and circle generation; as well as a WINDOW command that divides the CRT display into four different plot areas.

KBASIC allows optional parameters in the PLOT X,Y,P instruction for you to specify the line type for a vector, or the pattern to be used for filling or complementing a rectangular area. 'PLOT X,Y,10' and 'PLOT X,Y,11' are used for drawing circles and arcs. These are generated very quickly using one degree increments. The STRING instruction is used to specify the vertical shift between characters in a text string, or the bit pattern in special characters.

-PLOT X,Y,1,(,L)- with an optional parameter L, draws vectors with a line type given by L. The line types are as follows:

-PLOT X, Y, 4(, Pattern(,Slant))- allows you to specify the pattern to be drawn within the rectangle. The Pattern is given with a number from 1 to 11. A Pattern of 1 to 7 give stripes corresponding to the above line types. Patterns 8-11 fill the area with different tones.

When no Slant is given, the stripes are vertical. A Slant of 1 gives stripes slanting at 45 degrees. A Slant of 2 gives stripes slanting at 135 degrees. The same area may be filled in using Slants 0, 1, and 2 for checkered patterns. Slant is not used for Patterns 8-11.

-PLOT X, Y, 7(, Pattern(,Slant))- is the same as above, but without a rectangular frame drawn around the filled in area.

-WINDOW N- specifies the quadrant in which the plotting is done. N takes a value of 0 to 4. The quadrants 1 to 4 refer to the upper right, upper left, lower left, and lower right sections of the CRT. The default case, where a full CRT is used, is restored by WINDOW 0.

All plots are made using the normal range of 255x511 raster points. When a Window 1 to 4 is specified, a frame is drawn about the area, and the drawing, at 1/2 scale, is plotted in the given area. A CLEAR command will subsequently erase only the given Window area.

-PLOT X, Y, 10, Radius(, Start angle, Arc): makes a circle or arc. X and Y give the center point. The radius is given in increments in the X direction. The default values for the Start angle and Arc are 0 and 360 degrees, respectively. The Arc is drawn counter clockwise beginning at the Start angle.

-PLOT X,Y,11, Radius(, Start angle ,Arc): is the same as the above, except that lines are drawn connecting the center point with the end points of the arc.

-STRING A$, X, Y, F, R: is the normal string instruction when R is 0, 90, 180, and 270. However, when R is any other number, then it gives the number of vertical increment shifts between each character in the string. R may be a plus or minus number.

It is also possible to define the ASCII decimal characters 95 and 96 in any format within a 7 x 8 matrix. The two characters may be used together to form a special character in a 7x16 matrix. When the STRING instruction is used with a null text string, then X specifies the row in the character where the bit pattern of Y is loaded.

STRING "", X, Y: is used 14 times to load the logo "PSI" into two graphic characters. CHR$(95) is defined by the first column of X,Y values, and CHR$(96) by the second column.

X	Y	X	Y	CHR$(95)	CHR$(96)
1	249	8	231	11111 1	111 111
2	134	9	18	1 11	1 1
3	134	10	2	1 11	1
4	249	11	226	11111 1	111 1
5	128	12	18	1	1 1
6	130	13	18	1 1	1 1
7	129	14	231	1 1	111 111

The logo can be drawn with a height factor of 2 as follows:

```
100 A$ = CHR$(95)+CHR$(96)
110 STRING A$, X, Y, 2
```

The down loading of special characters should be only done when using Window 0. Characters can then be drawn in any window area.

CHR$(94) gives a degree symbol instead of the Greek psi.

Commands

LOAD PROGRAM.BAS	Load and compile PROGRAM.BAS
LOAD PROGRAM	Load PROGRAM.BAC
CHAIN SUBPROGRAM.BAS	Load and compile SUBPROGRAM.BAS
RUN (line no. optional)	Execute PROGRAM.BAC
CONT (line no. optional)	Continue execution after a STOP
KOS (command optional)	Execute KOS command

The compiler first replaces label references with line numbers. Variable names are then replaced by memory addresses. Wherever possible, lines are merged together. REM statements and labels are removed, and all jump references are changed to memory addresses.

During compilation, the line numbers are displayed as they are being handled. If an error occurs, a warning is given and, depending on the error, compilation will either stop or continue. Line numbers are shown with their memory addresses. This is done to help debug a program. If a run-time error occurs, then the error type is displayed with the file address of the affected line. The compiler information is output on channel #2 so that it may be obtained either on the CRT or on a printer.

After a .BAS type program is loaded and successfully compiled, it is written back on the disk as a .BAC type program.

The command LOAD PRIME.BAS gives the following output:

```
**************************************************************************

Labels
120

Line nos.
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150

Deleted line no. - file address
 40-16424      50-16424      60-16451      70-16451      80-16462
 100-16506     110-16538     130-16558     140-16600     150-16620

Remaining line no. - file address
 10-16390      20-16394      30-16398      90-16480      120-16551

Program compiled
Tue,  27 Mar 1985
PRIME.BAC

**************************************************************************
```

The space required by a compiled program is independent of the number of characters used in the Basic program for variable names or labels. The compiled program uses three bytes for a variable name. The statement 'A=B' takes three bytes in KBASIC and seven bytes in BASKOM. However, a statement such as 'XCOORD%=YCOORD%' takes 15 bytes in KBASIC, and still take seven bytes in KBASIC. The elimination of REM statements and labels, and the combining of program lines lessens the size of a compiled program.

Variables is a compiled program have relocatable addresses. The exact address depends upon the amount of space available in memory at the time BASKOM or BASRUN, and the compiled program are loaded.

In general, all programs written with Psi-BASIC or KBASIC can be compiled. However, The DIM statement must be used with a numeric constant for the index. Variables cannot be used as DIM subscripts. In the line number sequence, the DIM statement must precede lines containing array variables. Before being compiled, Psi-BASIC programs should be run first with KBASIC inorder to make sure that there are no incompatibilities.

BASKOM uses integer variables more efficiently than does KBASIC. SIN/COS functions are computed especially fast when arguments are integer variables, arguments are then assumed to be in degree units. Statements such as: IF I% THEN, IF I%=0 THEN, and IF I%<>0 THEN, with or without the ++ and -- operators, also run especially fast. Computation is also faster when integer variables are used for array subscripts and in simple arithmetic calculations. Statements with mixed operations are done much more efficiently when multiplications precede additions and subtractions.

'A% = B% * C% + D%' is preferable to 'A% = D% + B% * C%'

An instruction equivalent to the DEFINT of MBASIC is provided by use of a REM statement in which the first character is a percent '%'. All capital letters, coming after the '%' and before a colon ':' in the REM statement, will define as integer variables, those real variables whose name begins with the given letter.

```
10 REM % IJK: Variables starting with I,J, and K are integers.
```

Inorder to use the CHAIN instruction in a compiled program, the main program must be first loaded and compiled with the LOAD command. The subprograms, which are to be used, are then loaded and compiled with the CHAIN command. 'CHAIN SUBPROG.BAS' compiles a program the same same way as 'LOAD PROG.BAS', however, with CHAIN, the variable list from the preceding compilations are used. A MERGE instruction cannot be used in a compiled program.

The LOAD and CHAIN commands can also be combined in a single command line, as follows:

```
LOAD  PROG.BAS, PROG1.BAS, PROGR2.BAS, PROGR3.BAS, PROG4.BAS
```

In this case, the PROG.BAS is loaded and compiled. The subprograms, separated by commas, are then each in turn loaded and compiled in the same way as if they were each preceded by a CHAIN command. It is generally best to define all variables in the main program, but it's not required to do so. However, no variable may be used unless it has been defined in a preceding subprogram.

In a program the KOS argument is a string or string variable. As a direct command, however, the argument is in the normal text form.

During execution of a program, the TRACE mode may be toggled on and off with CNTRL-T inorder to follow the progress of the program. Due to the combining of lines in a compiled program, only the remaining line numbers are displayed. In general, these are the lines to which a jump may be made. The STEP mode is not used. TRACE output is on channel #2 when in graphic mode.

BASRUN is a subset of BASKOM that can only execute the commands RUN, CONTinue, and KOS. Any other input is interpreted as a .BAC type program name, which will be loaded and run immediately. The command BASRUN "PROG" entered from KOS can be used to load the the programs and start execution of PROG.BAC.

In general it is better to use BASRUN instead of BASKOM for running a program. BASRUN runs somewhat faster, uses less memory, and gives more complete error messages.

Following are execution times under KOS-5 for the PRIME program:

PSI-BASIC	40 sec.	using IF/THEN in place of WHILE/WEND
MBASIC	22.6 sec.	
KBASIC	20.4 sec.	using line numbers, and 21.4 using labels
BASKOM	7.3 sec.	with integers (Line 20), and 11.7 with reals

Error messages are similar to those of KBASIC, however, in place of a line number, the file address is given along with the ERM number. The information displayed at compilation indicates the affected line number. All variables are assigned an initial value of zero.

Memory overflow could occur if there are more driver programs in memory during program execution than there were when the program was compiled. This may show as a syntax error at file address 1.

ERM	Error message	Likely reason for error
59	bad I/O file no.	Illegal channel number used
60	file not found	File inexistent or faulty disk
62	integer overflow	LINELEN value too large or too small
64	syntax	Missing comma in program line
65	stack overflow	Too many digits in INPUT value
66	stack overflow	More than 8 FOR/NEXT loops in use
67	file not closed	File in given channel still open
68	repeated DIM	Array already dimensioned (warning)
71	loops?	ENDed while in subroutine or loop
72	integer overflow	Incorrect number used
74	FOR/NEXT index?	Mismatch in FOR/NEXT index variable
75	GOSUB() parameters	Inconsistent no. of arg. in GOSUB( )
76	syntax	Incorrect character in USING format
77	insufficient memory	No. of variables is too large
79	math. overflow	Division by 0, or number > 1E+127
80	array > DIM	Subscript greater than DIM value
81	SQR or LOG arg.	Illegal arg. in SQR or LOG function
82	loops?	Illegal GOSUB, WHILE, or REPEAT loop
83	syntax	Incorrect operator or data
84	stack overflow	Too many GOSUB, REPEAT, or WHILE loops
85	variable type	Mismatch in variable or data types
86	variable type	String function or variable expected
87	end of file/data	No more INPUT, READ, or FIND data
88	integer overflow	Incorrect value used in program

KOS system messages

96	invalid parameter	KOS I-O error
98	I-O channel not ready	No diskette in unit
99	I-O channel not ready	IODC not invoked or channel not open
100	data transfer	Faulty diskette or wrong channel no.
102	file protected	Diskette write protected
104	medium occupied	Diskette is full
105	media/memory occupied	Program too large, cannot be loaded
109	file protected	File write/erase protected
110	media/memory occupied	Faulty block allocation on diskette

Listings of sample PRIME.BAS program using PLIST and BLIST

```
10    ! CALCULATE PRIME NUMBER
20    ! %FSCIPK: INTEGER VARIABLES FOR BASKOM
30  DIM FLAGS(1000):SIZE=1000:COUNT=0
40   ! SET ALL FLAGS TO 1
50  FOR I=0 TO SIZE:FLAGS(I)=1: NEXT I
60    ! DO LOOP UNTIL ALL FLAGS ARE ZERO
70  FOR I=0 TO SIZE
80  IF FLAGS(I)=0 THEN 'NEXT
90  PRIME=I+I+3:K=I+PRIME
100  WHILE K=)SIZE:FLAGS(K)=0:K=K+PRIME
110  WEND:COUNT=COUNT++
120  'NEXT: NEXT
130  PRINT "PRIME= ";PRIME,"COUNT= ";COUNT,"SIZE= ";SIZE
140  PRINT TIMER(0)/50;" SECONDS"
150  END
```

```
10      ! CALCULATE PRIME NUMBER
20      ! %FSCIPK:  INTEGER VARIABLES FOR BASKOM
30    dim FLAGS(1000)
      SIZE = 1000
      COUNT = 0
40      ! SET ALL FLAGS TO 1
50    for I = 0 to SIZE
        FLAGS(I) = 1
      next I
60      ! DO LOOP UNTIL ALL FLAGS ARE ZERO
70    for I = 0 to SIZE
80      if FLAGS(I) = 0
           then 'NEXT

90       PRIME = I+I+3
         K = I+PRIME
100       while K ) SIZE
            FLAGS(K) = 0
            K = K+PRIME
110       wend
          COUNT = COUNT++
120      'NEXT
        next I
130    print "PRIME= ";PRIME, "COUNT= ";COUNT, "SIZE= ";SIZE
140    print timer(0)/50;" SECONDS"
150    end
```

Display of defined program variables using VLIST

```
        FL()     SI       CO       I        PR       K
```

List of KBASIC Commands and Instructions

Commands

```
ALOAD    (mn:)filename(.type)                    .BSC type
ASAVE    (mn:)filename(.type)                    .BSC type
AUTO     (B=begin line)(S=line spacing)          Default values: B=10  S=10
BLIST    (line1-line2)                           Formatted program listing
CONT     (line no.)                              Continue program execution
DELETE   (line1-line2)                           Delete given line
FE       line no.                                FE, FETCH line for editing
INFO     (mn:)filename                           Display .INF type file
KOS      ("function")                            Execute KOS fnct. or exit
LI,LIST  (line1-line2)                           List program lines
LI       Variable                                List lines with Variable
LI       #line no.                               List lines containing no.
LI       Variable1/Variable2                     Change Var1 name to Var2
LOAD     (mn:)filename(.type)                    .BAS type
NEW                                              Erase program from memory
PLIST    (line1-line2)                           Output list to channel #4
RL                                               Relist last 'LI line1-line2'
RNB      (line1)(-line2)(B=begin line)(S=line spacing)   Renumber
RUN      (line no.)                              Execute program
SAVE     (mn:)filename(.type)                    .BAS type
VKEY     "String"                                Define input via 'v' key
VLIST                                            List defined variables
```

Instructions

```
CALL     adr(,p1)(,p2)(,...)(,pn)                GOSUB to compiled subroutine
CHAIN    "filename"                              LOAD new program module
CLEAR                                            Clear graphic window area
CLOSE    #var:                                   Close file or graphic mode
DATA     val1,val2,.....,valn                    Data for READ statement
DEGREE                                           Use degrees with trig functs.
DEL      #var:                                   Delete open file
DIM      var1(1(,m,...,n))(,var2....)            Define variable array size
DOUBLE                                           13 digit reals (default)
ELSE     Instruction or line no.                 Error test for I-O operation
END                                              Terminate program execution
FIND     #var:var$                               Find record containing var$
FOR var=v1/TO v2/(STEP v3)/NEXT (var)            FOR/NEXT loop
GET      (#var:)var$                             Input 1 ASCII character
GET      #var:var%                               Input 2 byte integer
GOSUB    line no./'label((p1,p2,...,pn))         Optional subr. parameters
GOTO     line no./'label                         Jump to line no. or 'label
IF       condition
         THEN statements or line no./'label      (condition true)
         (ELSE statements or line no./'label)    (condition false)
INPUT    (#var:)var1(,var2,...varN)              Read in ASCII characters
INV      x,y                                     Invert graphic point x,y
```

```
KOS      ("function")                    Execute KOS function
(LET)
LINELEN I=input line length              20 to 132 (default case)
LINELEN O=output line length             20 to 132, or * (infinite)
MERGE    "filename"(,"line1-line2")      ALOAD,  after deleting lines
ON var  GOTO/GOSUB line1,line2, etc.     var is integer
ON ERROR instruction                     GOTO, GOSUB, or RETURN
ON INTR instruction                      GOTO, GOSUB, RETURN, SET, RES
OPEN     #var:"file or driver name"      Default file type is .DAT
OUT      port, data                      Output data byte to port
PLOT     x,y,p                           Line p=0-3; rect. p=4-9
PLOT     x,y,1,linetype                  Dashed line, linetype 1-11
PLOT     x,y,4(,pattern(,slant))         Frame (and fill) rect. area
PLOT     x,y,7(,pattern(,slant))         Fill rectangular area
PLOT     x,y,10,r(,ang,arc)              Circle or arc
PLOT     x,y,11,r,ang,arc                Sector
POKE     adr, p1(,p2,...pn)              POKE in sucessive adresses
PRINT    (#var:) var1,.....,varn         Ouput ASCII characters
PRINT USING "format", var1,....,varn     Formatted ASCII output
RADIAN                                   Radians for trig fn.(default)
(RANDOM)                                 Not used, acts as NO OP
READ     var1,.....,varn                 Read DATA line
READ     #var:var1.....,varn             Input binary data from file
RECORD   #var: record no. or '*'         Position file to record no.
REM      Text                            Shown as '!' in listings
REPEAT                                   Begin of REPEAT/UNTIL loop
RES      x,y                             Reset graphic point x,y
RESTORE                                  Reset DATA pointer
RETURN   ((p1,...,pn))                   Return (optional parameters)
SET      x,y                             Set graphic point x,y
SINGLE                                   Use 7 digits for reals
STEP                                     Set single step TRACE mode
STOP                                     Stop program execution
STRING   var$,x,y(,factor,rotation)      Text output in graphic mode
SWAP     var1, var2                      Exchange variable values
TONE     m, n                            Acoustic tone 'm', 'n' times
TRACE                                    Display lines during RUN
TRACEOFF                                 Disable Trace mode
UNTIL condition                          REPEAT loop until cond. true
USING    "format"                        Symbols: "# + - * . , ^ &"
WHILE condition / WEND                   Do loop while condition true
WINDOW   m                               1-4 quadrant, or 0 (default)
WRITE    #var: var1,.....,varn           Binary output to file
'        'label                          Label in place of line number
```

Functions

```
ABS(var)                                 Absolute value
ATN(arg)                                 Arctangent, (rad. or deg.)
COS(arg)                                 Cosine, (rad. or deg.)
ERM(0)                                   Error code
EXP(arg)                                 e raised to power of arg.
FRAC(var)                                Fractional part of var.
FRE(0)                                   No. of free program bytes
```

IN(port)	Input byte from port
INT(var)	Integer part of var.
LOG(arg)	Natural log of arg.
PEEK(adr)	Read byte from memory adr.
POINT(x,y)	Test if graphic point is set
REC(0)	Current record no.
RND(0)	Random no. between 0 and 1
SGN(var)	1, -1, or 0 depending on var.
SIN(arg)	Sine, (rad. or deg.)
SQR(arg)	Square root
TAB(n)	No. of spaces to PRINT
TAN(arg)	Tangent, (rad. or deg.)
TIMER(arg)	20 msec. timer; 0-65535 count
VPTR(var)	Stack address of var.
'label(args) / RETURN(var)	Programmed numeric function

String Functions

ASC(var$)	numeric value of ASCII char.
CHR$(var)	ASCII equiv. of var. value
DAY$	Day and Date
LEFT$(var$, n)	n bytes from left of var$
LEN(var$)	No. of bytes in var$
MID$(var$, m, n)	n bytes from m+1 on
POS(var1$,var2$)	Position of var2$ in var1$
RIGHT$(var$, n)	n bytes from right of var$
STR$(var, ("format"))	var. value in string form
VAL(var$)	Give var$ in numeric form
"c"(n)	Repeats char. "c" n times

Data types

Reals:	Floating point BCD numbers in 8 bytes DOUBLE precision has 13 digits (normal case) SINGLE precision has 7 digits
Integers:	Fixed point in 2 bytes
Strings:	ASCII form, 1 byte per character
Hexadecimal:	&0 to &FFFF, number preceded by symbol '&'

Variables

Reals:	A, AZ, Z, A', A0, A0', Z9, A(n), X1.POINT
Integers:	A%, AZ%, Z%, A'%, A0%, A0'%, Z9%, A%(n), Y1.COORD
Strings:	A$, AZ$, Z$, A'$, A0$, A0'$, Z9$, A$(n), VA$.VALUE

Variable names can be any length, but only the first 2 characters and a tic mark are used. Characters include upper case letters, numbers, and decimal points. The first character in a name must be a letter. The second character may be a letter, number, or tic mark.

Contents of KBASIC Diskette

KBASIC	COM	English version
KBASICG	COM	German version
BASKOM	COM	
BASRUN	COM	
GRAF	OBJ	$GRAF Driver PSI-80 KOS 4/5
GRAF82	OBJ	$GRAF Driver PSI-82 KOS 4/5
KOS6	SYS	$GRAF Driver PSI-98/908/980 KOS 6
HELP	INF	File for use with INFO HELP command
KBASIC1	INF	Program description 1
KBASIC2	INF	Program description 2
GRAFDEMO	BAS	Demo programs
MAGIC	BAS	
TONEDEMO	BAS	
CALLDEMO	BAS	
PRIME	BAS	

For KOS 4/5 systems it is only necessary to activate $GRAF in place of $GRAP. For KOS 6 systems it is necessary to replace the KOS6.SYS file on a system disk with the one from the KBASIC diskette. This is done by first using the DEFP command to reset all the flags on the original KOS6.SYS file. Then you can use either MOVE or COPY to replace the file. After replacement of the KOS6.SYS, you should use IL P=* to check that it is located within the first 32 files on the system disk. The $GRAF driver is activated the next time that the system is booted up.

$GRAF can be used with Psi-BASIC under KOS 4/5 using standard Psi-BASIC graphic instructions. Under KOS 6, however, there must be a STRING instruction used before any PLOT instructions are used. This is to initialize a buffer for x,y values. The first program line is:

```
10 OPEN #9:"$GRAF": STRING "",0,0: REM Psi-BASIC (KOS 6)
```