[illegible]

0. INDEX

N°	DATE	SUBJECT
0000/rev 2	84.07.23	Index
0001	84.06.22	Introduction
0002	84.06.28	BASIC version 1.1
0003	84.06.22	Joystick (2 action buttons)
0004	84.06.22	Screen editing/1
0005	84.06.22	INPUT command
0006	84.06.22	NEW command
0007	84.06.22	AUTO command
0008	84.06.28	Sumcheck
0009	84.06.22	Version 1.0 TB's
* 0010	84.06.22	CLEAR command/1
* 0011	84.06.22	SETET/SETEG commands
* 0012	84.06.22	CLOADA command
* 0013	84.06.22	RUN command
* 0014	84.06.22	Piracy protection/1
* 0015	84.06.28	BASIC entry points/1
* 0016	84.06.28	Screen transfer
* 0017	84.06.29	Input streams
* 0018	84.06.29	Output streams
* 0019	84.06.29	BASIC tokens
* 0020	84.07.05	FRE(X$) function
* 0021	84.07.05	CLEAR command/2
0022	84.07.09	Screen editing/2
* 0023	84.07.12	PRINT/LPRINT USING command
* 0024	84.07.12	LPRINT command
* 0025	84.07.23	Joystick interface
* 0026	84.07.23	Machine-code programs on cassette
* 0027	84.07.23	PLAY command
* 0028	84.07.23	CALL command
* 0029	84.07.29	CONT command
* 0030	84.09.11	CSAVE/CLOAD commands

N°	DATE	SUBJECT
* 0031	84.09.11	RESET
* 0032	84.09.11	BASIC's line format
* 0033	84.09.11	BASIC's entry points/2
* 0034	84.09.11	Memory map
* 0035	84.09.11	Piracy protection
* 0036	84.10.12	CLEAR command/3
* 0037	84.10.12	BASIC's hooks

* (also BASIC version 1.0)

N°	DATE	SUBJECT
* 0031	84.09.11	RESET
* 0032	84.09.11	BASIC's line format
* 0033	84.09.11	BASIC's entry points/2
* 0034	84.09.11	Memory map
* 0035	84.09.11	Piracy protection
* 0036	84.10.12	CLEAR command/3
* 0037	84.10.12	BASIC's hooks
* 0038	84.10.19	FOR...NEXT command
* 0039	84.10.19	CALL command/2
* 0040	84.11.09	STORE/SCREEN/DISPLAY commands
* 0041	84.11.09	Interrupts
* 0042	84.11.09	BASIC's current line position

* (also BASIC version 1.0)

1. INTRODUCTION

1.1 The purpose of these Technical Bulletins is to publish, in a referenced format, information on features, bugs and programming using BASIC on the VG 5000.

1.2 The reader is assumed to be familiar with other publications relating to the VG 5000. Where necessary, reference will be made to these other information sources.

1.3 This Bulletin series refers to BASIC version 1.1 (ROM reference C 11)- no further production version is expected.

1.4 The information process should be two-way , if the reader knows of any feature, bug or programming technique that is not covered by a Bulletin, please write to the following address :

M. P. DURAND
Division III
Compagnie française Philips
50 avenue Montaigne
75380 PARIS CEDEX 08

Be sure to enclose a concise description of the item together with a short program that demonstrates the point being made.

2. BASIC VERSION 1.1

2.1 All changes have been made by patches. The entry points for all subroutines are unaltered, with one exception :

MODULE	: bimisc
ENTRY	: scrath
ADDRESS	: 3DCE (was 2ED8)

As this entry point is only used by the NEW command, no problems should be found. The old entry point at address hex 2ED8 is still available.

2.2 Free space in the ROM has altered.

001FFE	0002	Free
003E3F	01CI	Free

2.3 The changes are transparent, and mainly improve the end-user interface. The only new programming feature is the addition of a second action-button to the ACTION (x) function.

2.4 The attached pages document in detail the patches that have been made.

PATCHES : version 1.1 from version 1.0

PATCH#	DETAILS
1.1/0000/00	Increases the version number to 1.1
1.1/0001/00	Corrects the hexadecimal number evaluation error (BASIC 1.0-TB 2.2)
1.1/0002/00	Corrects the hexadecimal number trailing space error (BASIC 1.0-TB 2.4)
1.1/0003/00	Corrects the screen column 39 character delete error (BASIC 1.0-TB 3.2)
1.1/0004/00	Stops dropping-out of AUTO mode on exact screen-width lines (BASIC 1.0-TB 3.3)
1.1/0005/00	
1.1/0005/01	Cancels AUTO mode if a NEW is executed (BASIC 1.0-TB 4.2)
1.1/0006/00	Corrects printer translation character hex F1 (BASIC 1.0-TB 5.2)
1.1/0007/00	Provides for 2 action button joysticks (BASIC 1.0-TB 7.1)
1.1/0008/00	
1.1/0008/01	
1.1/0008/02	Corrects the multiple INPUT command data entry for second and subsequent variables (BASIC 1.0-TB 9.2)
1.1/0009/00	Stops the INPUT command turning off the PAGE command (BASIC 1.0-TB 9.4)
1.1/0010/00	
1.1/0010/01	Improves the handling of overlength BASIC lines by the screen editor (BASIC 1.0-TB 13.1)
1.1/0011/00	Corrects the LOAD command start line number error (BASIC 1.0-TB 15.1)

PATCH#	DETAILS
1.1/0012/00 1.1/0012/01	Provides an optional parameter for the NEW command
1.1/0013/00 1.1/0014/00 1.1/0015/00	Are not incorporated
1.1/0016/00	Corrects the POKE command value to the positive integer range 0-255 only (BASIC 1.0-TB 16.2)
1.1/0017/00	Not incorporated
1.1/0018/00 1.1/0018/01 1.1/0018/02 1.1/0018/03 1.1/0018/04 1.1/0018/05	Improves the AUTO command to warn that the generated line number is already used in the program (BASIC 1.0-TB 17.1)
1.1/0019/00	Prevents the AUTO command from going past line number 65529 (BASIC 1.0-TB 17.2)
1.1/0020/00 1.1/0020/01 1.1/0020/02 1.1/0020/03 1.1/0020/04	Prevents the "insert line" key from operating during an INPUT command, and stops the screen editor from automatically issuing an "insert line". Also preserves PAGE command status during INPUT commands (BASIC 1.0-TB 18.1/2) (BASIC 1.0-TB 10.2)
1.1/0021/00 1.1/0021/01	Disables an "insert line" generation during "insert character" in an INPUT command (BASIC 1.0-TB 18.3)
1.1/0022/00	Built in sumcheck

1.1/0023/00	
1.1/0023/01	Prevents LIST keys from operating during "allflg" protection
1.1/0024/00	
1.1/0024/01	Improves screen transfer time to $<$40 ms and improves keyboard response/repeat rate.

3. JOYSTICK (2 ACTION BUTTONS)

3.1 Two action button joysticks are now supported.

3.2 In BASIC, the ACTION (x) function is unchanged. The value returned is extended as follows :

0 = no button pressed
1 = action button 1 pressed (or < SPACE >)
2 = action button 2 pressed
3 = both action buttons pressed

3.3 In a call to the BIOS subroutine "stka" the value is returned in register A :

A = 0 no button pressed
A = 1 for action button 1
A = 2 for action button 2
A = 3 for both action buttons

4. SCREEN EDITING

4.1 If a line is entered into BASIC that is exactly the maximum line length or longer, a warning "beep" is sounded to warn the programmer.

4.2 An overlength line will be truncated, but the cursor will be corectly repositioned for the next line entry.

5. INPUT COMMAND

5.1 All screen editor features that can cause a partial screen scroll are disabled.

5.2 If a PAGE command has been issued, the screen will not scroll, even if the INPUT command is executed on screen line 25 (bottom of the screen).

5.3 The user can still "roam" over the screen using the cursor keys, and the CTRL input of BASIC reserved words is still active. Multiple input lines for string variables are still available.

6. NEW COMMAND

6.1 An optional parameter can be used with the NEW command :

```
NEW nnnnn
```

nnnnn represents a line number.

6.2 On execution, only lines nnnnn and upwards are erased from the program. For example :

```
NEW 1000
```

- would erase all user program lines from a BASICODE-2 program.

6.3 The command can be used in direct, or indirect modes. If used within a program, it will be executed and control returned to BASIC's direct mode. All variables of the program will also be erased.

7. NEW COMMAND

7.1 If the AUTO command generates a line number that already exists in the user's program, a warning "diamond" is printed next to the line number.

7.2 The "diamond" character is transparent, that is, it will never be included in a BASIC program line. It is never necessary to "space" over it.

7.3 The "diamond" is produced by a specially preprogrammed character. It occupies the code hex 20 in the ET character set. Like (..), if the ET character set is reprogrammed the (◆) may disappear, and be replaced with a new character. (◆) is only programmed at power on and RESET.

7.4 If the AUTO command generates a line number greater than 65529, it terminates.

8. SUMCHECK

8.1 Bytes 000026/000027 are used for a built-in sumcheck :

000026	39	Sumcheck remainder
000027	FC	Sumcheck partial sum

8.2 A 16-bit addition of the ROM will produce the value :

FCFC hex

Note that bits 0-7 = 8-15 = address 27 value. The summing algorithm used is the same as DATA I/O EPROM programmers, so a copy operation of the ROM should also produce the sumcheck FCFC.

9. TECHNICAL BULLETINS - VERSION 1.0

9.1 The BASIC version 1.0 Bulletins that also apply to version 1.1 are as follows :

0006	84.06.04	Version number
0007	84.06.04	Joystick (7.2 onwards)
0008/rev 1	84.06.14	BASIC text relocation
0011	84.06.12	Screen control characters
0012	84.06.12	VARPTR(X$) function
0013	84.06.12	Screen editing/2 (13.2 onwards)
0014	84.06.14	RESET

10. CLEAR COMMAND

10.1 This command, if used, should be the first one in the program.

10.2 As well as allocating string variable space and memory available, it also resets BASIC's stack. The context of IF ... NEXT, GOSUB... RETURN are lost and SCROLL is enabled.

(also BASIC version 1.0)

11. SETET/SETEG COMMANDS

11.1 The character shape programming string is directly sent to the video system. The VARPTR(X$) emulation will not work (BASIC 1.0-TB 12).

11.2 If dynamic reprogramming of character shapes is required, a USR(X) subroutine should be written to support the "setsext" facility in the BIOS (VG 5000 TECHNICAL MANUAL 6 BIOS SECTION page 1).

(also BASIC version 1.0)

12. CLOADA COMMAND

12.1 CLOADA computes its initial load point relative to the pointer "vartab". Usually "vartab" points to the ram location just beyond the current user BASIC text.

12.2 When CLOADA has completed, it rechains the resultant BASIC text file, ignoring line number order, and resets "vartab" to point to the new ram location just beyond the current user BASIC text.

12.3 Programs that use CLOADA to load subroutines, or alter the value of "vartab" must take these points into account.

(also BASIC version 1.0)

13. RUN COMMAND

13.1 The RUN command (with or without parameter) carries out the following :

CLEAR
RESTORE
BASIC STACK initialisation
Printer deinitialisation

13.2 Any program that requires a different memory configuration, in particular, changes to variable pointers, must carry out these changes in the first lines of the program. Beware of evaluating variables that the pointers are then changed for !

(also BASIC version 1.0)

14. PIRACY PROTECTION

14.1 Most piracy takes place at commercial level using high-speed "back-to-back" cassette copying machines : This is difficult to prevent by technical means.

14.2 Two other kinds of piracy take place :

- casual copying (e.g : computer clubs amongst their numbers),

- intellectual copying (e.g : "cloning" of games for the same, or other computers).

14.3 For a program written in BASIC, two techniques can be used. Together, they are virtually impossible to break.

14.4 A byte called "allflg" is located at address hex&"4889". If this byte is other than zero, then :

- SHIFT/STOP is disabled
- LIST key is disabled (BASIC version 1.1 only)
- any direct command will scratch memory by executing a CALL 0.

The "allflg" can be set to a value PRIOR to CSAVEing a program. This value is stored in the cassette file table, byte 13. On a subsequent CLOAD, the value in the file table is OR'd with the "allflg". Thus subsequent CLOAD's can be accumulated in "allflg" if required. An auto-execution cassette program will always run, even if the start line number is overriden as 0, if "allflg" is not zero.

14.5 A known set of values can be read from memory on commencing execution. These values can be stored in program line 0. This will prevent a program from CLOADAing the protected program, reseting the "allflg" and ENDing.

14.6 An example of protection is given below. Note that the first token of a BASIC program will usually be found in address hex&"4A00".

Program :

```
0 REM#
10 PRINT "Running OK!  ";
20 GOTO 10
12345 IF  66 <> PEEK (&"4889") THEN CALL 0
12350 IF 142 <> PEEK (&"4A00") THEN CALL 0
12355 IF  35 <> PEEK (&"4A01") THEN CALL 0
12360 IF   0 <> PEEK (&"4A02") THEN CALL 0
12365 GOTO 10
```

To CSAVE :

```
CSAVEL:POKE&"4889",66:CSAVE"Break",12345:POKE &"4889",0 <RET>
```

14.7 Also, this scheme is easy to adapt for machine-code.

(Also BASIC version 1.0)

15. BASIC ENTRY POINTS

15.1 To enter/reenter BASIC, the following entry points are provided :

Name Address	Description Parameters
reset 0000	Restart BASIC from power on condition. Resets all of BASIC's variables and calls any ROM it can find. Reinitialises all of memory and I/O (except for memory bank switching). entry none exit none modifies all
warmgo 0033	Same effect as pressing CTRL/RESET. Does not vector via "nmihk", so is non-maskable. Resets video colours/mode and stack and plays the "jingle". Reinitialises I/O (except for memory bank switching). programs are preserved. entry none exit none modifies all
reset2 101C	Restarts BASIC from power on condition, but with the memory size specified by register pair HL. No memory test is done. Resets all of BASIC's variables and calls any "ROM" it can find. Reinitialises I/O (except for memory bank switching). entry HL = address of highest byte of memory exit none modifies all

resign 10AB	Restarts BASIC from the sign on message. Calls any "ROM" it can find. Programs are preserved.	
	entry	none
	exit	none
	modifies	all
ready 228D	Closes line-printer file, returns cursor to column 1 on next line if not already there and prints "Ok!". Drops into "main".	
	entry	none
	exit	none
	modifies	all
main 2299	Looks for a direct/defered command line from the current input stream.	
	entry	none
	exit	none
	modifies	all

All the above entry points should be jumped to.

fini 2328	Does a CLEAR and reinitialises the stack. Re-links the program lines and goes to main.	
	entry	none
	exit	none
	modifies	all

This entry point can be called or jumped to.

newstt 24EA	The address of this routine is left on the stack when a statement is executed. A RETurn will come back to here. HL must be set to point to the end of the previous statement (: or NUL)	
	entry	HL = statement address
	exit	none
	modifies	all

16. SCREEN TRANSFER

16.1 A small bug exists in BASIC version 1.0 that can cause momentary screen-blanking of part of the display.

16.2 As part of the correction, the screen transfer speed has been increased from > 40 ms to < 40 ms. Depending on the amount of processing overhead, screen updating can be up to 50 % faster.

16.3 This correction has also improved the keyboard repeat rate and response.

(also BASIC version 1.0)

17. INPUT STREAMS

17.1 BASIC provides for two imput streams as selected by the variable "getflg" at address hex 4870 (decimal 18544) :

```
      0 = terminal (keyboard)
    255 = cassette (ASCII file)
```

If using "inlhk" this variable should be examined to determine which stream was requested.

17.2 The ASCII file can be stored anywhere in memory. A data pointer to the ASCII file is held in locations hex 4828/4829 (decimal 18472/18473). As it is always incremented prior to use, the value of the pointer should be the ASCII file address - 1.

17.3 To switch to the cassette stream, set up the data pointer then POKE "getflg" with 255.

17.4 The input cassette stream can contain a mixture of direct (no line numbers) or deferred (line number) command lines. The lines are in ASCII (i.e : untokenised) and must be $<$ 127 bytes long. Each line will be executed when cariage return (hex OD) is encountered. The stream can be any length. If a processing error occurs (e.g : "Syntax error") processing continues with the next line in the file.

17.5 ASCII ETX (hex 03) terminates the ASCII file and resets "getflg" to 0. The final two bytes in the file are therefore hex OD 03.

17.6 If cassette input is processed during an ASCII file the data pointer will be lost. Cassette output is permitted.

17.7 See the next page for an example of executing a direct command from an ASCII file.

(also BASIC version 1.0)

Demonstration of a DIRECT command via BASIC

```
10 REM ************************
20 REM *                      *
30 REM *  TO EXECUTE A DIRECT  *
40 REM *                      *
50 REM ************************
60 REM
70 RESTORE:REM Reset the data pointer
80 FOR A=18600 TO 18699:REM Use some of BUF
90 READ D:POKE A,D:REM Poke it away to BUF
100 IF D<>3 THEN NEXT A:REM Stop on a 3
110 POKE 18544,255:REM Redirect input stream
120 A=18472:POKE A,&"A7":POKE A+1,&"48":END:REM Point to 18600-1 and END
130 DATA 82,85,78,49,48,48,48,13,76,73,83,84,13,3:REM Put what you like here - 3
 closes the stream
1000 FOR I=1 TO 10
1010 PRINT "Hello !!"
1020 NEXT I
1030 END
```

18. OUTPUT STREAMS

18.1 BASIC provides for three output streams as selected by the variable "prtflg" at address hex 486F (decimal 18543) :

```
  0 = terminal (screen)
  1 = printer
255 = cassette (ASCII file)
```

If using "outhk" this variable should be examined to determine which stream was requested.

18.2 "Carriage return" sequences are generated by BASIC (for example by PRINT commands). Each sequence makes one CALL via "crdhk".
Again, if using "crdhk" to trap these CALL's the "prtflg" variable should be examined to determine which stream was requested.

18.3 The graphics printer driver for VW 0010/VW 0020 printers uses "outhk".
If "outhk" is already in use, the users routine CALL is preserved, and executed prior to the graphics printer routines.

(also BASIC version 1.0)

19. BASIC TOKENS

19.1 Three lists exist in the BASIC :

Reserve word
Token
Transfer Address

Reserve words start at address hex 209E. Tokens are implied by the position of the Reserve Word in the list. The first word is hex 80 (decimal 128) and the last word is hex DF (decimal 223). Transfer Addresses are 16 bit words that start in address hex 2000 (decimal 8192).

19.2 See the next page for a small program that will print the Reserve word list and the associated Tokens.

(also BASIC version 1.0)

TOKENS LISTING PROGRAM FOR VG5000

```
10 REM ***************************
20 REM *                         *
30 REM *   RESERVE WORD LISTING   *
40 REM *                         *
50 REM ***************************
60 :
70 :
80 PRINT CHR$(31);"RESERVE WORD LIST"
90 PRINT"==================":PRINT
100 PRINT"(token)    (word)"
110 ADDR=&"209E":COUNT=128:GOTO 160
120 :
130 :
140 A$=CHR$(PEEK(ADDR))
150 IF A$>"z"THEN PRINT" ";COUNT;"        ";ALPHA$;CHR$(4):COUNT=COUNT+1:ALPHA$=""
160 IF COUNT=224 THEN GOTO 210
170 IF A$>"z" AND 23=POS(255) THEN INPUT Z$:PRINT CHR$(9);CHR$(4);:CURSORY 4
180 A$=CHR$(PEEK(ADDR)AND127)
190 ALPHA$=ALPHA$+A$
200 ADDR=ADDR+1:GOTO 140
210 IF 23>POS(255) THEN PRINT CHR$(4):GOTO 210
220 END
```

20. FRE(X$) FUNCTION

20.1 The documented use of the FRE function is to return the amount of free memory space below BASIC's stack. This memory space is used for :

BASIC program text
Numeric variables
Arrays (numeric and string)

A dummy numeric expression is used, such as FRE(A) or FRE(0). The value returned is directly related to the second parameter of the CLEAR command.

20.2 An undocumented used of the FRE function is to return the amount of free memory space above BASIC's stack (but within memory size as set by the BASIC variable "memsiz"). This memory space space is used for string variables.
A dummy string expression is used, such as FRE(A$) or FRE(" "). The value returned is directly related to the first parameter of the CLEAR command.

(also BASIC version 1.0)

21. CLEAR COMMAND

21.1 The current value remaining of the first parameter can be obtained from :

```
?FRE(" ")              <RET>
```

21.2 The last 3 bytes of memory are located above "memsiz" (for any configuration of RAM). As well as using CLEAR to decrease "memsiz", it can also increase "memsiz" without losing a user program.

21.3 To obtain maximum memory in a standard VG5000 console enter :

```
CLEAR50, 32767 <RET>
```

The statement :

```
? FRE(0)
```

will now return :

```
13761
```

21.4 If the "memsiz" parameter is set too low, the command may be accepted, but due to lack of space all subsequent commands may report "out of memory". A fresh power on is required to restart BASIC.

(also BASIC version 1.0)

22. SCREEN EDITING

22.1 If the < LIST > key is used to display BASIC program text for editing, scroll is locked until the < RET > key is pressed for the first time.

22.2 If the line to be edited is the last line on the screen, pressing < RET> will alter the text, but no scroll will occur unless < RET> is pressed a second time. If this line has continuation lines the computer may lock-up with an asynchronously flashing cursor. CTRL/RESET will be necessary to recover from the problem. In this case the line will be unchanged.

22.3 It is not recommended to edit the last line on the screen produced by the < LIST> key. it may have a hidden continuation that could be erased without the programmer being aware of it.

(also see TB 4)

23. PRINT/LPRINT USING COMMAND

23.1 This powerful print field edit command is not provided. However, a small routine written in BASIC can provide simple editing of numeric fields with a value between -1E6 and + 1E6.

23.2 The routine is listed on the next page. To demonstrate the routine, enter the following lines :

```
1000 INIT:PAGE
1010 L=POS(255):CURSORY23
1020 INPUT"Enter value, column width, decimal places";V,W,X
1030 GOSUB 170
1040 CURSORYL
1050 PRINT,Z$:GOTO 1010
```

Now enter :

```
RUN 1000 <RET>
```

23.3 The primary use of this routine is to print numeric values in columns with aligned decimal places. Z$ will consist of "*'s" if the value is too large for the column width requested.

(also BASIC version 1.0)

NUMERIC FIELD EDITOR

```
10 REM *******************
20 REM *                 *
30 REM *  NUMERIC EDITOR  *
40 REM *                 *
50 REM *******************
60 :
70 :
80 REM This subroutine will edit a numeric field acoording to parameters:
90 : REM V =Value to be edited (<1E6)
100 :REM W =Width of edited field
110 :REM X =Number of significant decimal places required
120 :REM Z$=Output field
130 :REM   - V W X are unchanged
140 :REM   - S T X$ Y are work variables
150 :
160 :
170 Y=ABS(V)+.5*10^-X:T=INT(Y):S=Y-T+1:Z$="":IFY=>1E6THEN 250
180 IFX=0THENX$="":GOTO 220
190 IFS=1THENX$=".":GOTO 210
200 X$=MID$(STR$(S),3,X+1)
210 IFLEN(X$)<X+1THENX$=X$+"0":GOTO 210
220 Z$=MID$(STR$(T),2)+X$:IFV<0ANDVAL(Z$)<>0THENZ$="-"+Z$
230 IFLEN(Z$)<W THENZ$=" "+Z$:GOTO 230
240 IFLEN(Z$)>W THENZ$=""
250 IFLEN(Z$)<W THENZ$=Z$+"*":GOTO 250
260 RETURN
270 :
280 :
290 REM (Acknowledgements to NOS-BASICODE Group)
```

24. LPRINT COMMAND

24.1 If using VW0010 and VW0020 printer, the VG 5000 characters between hex 11 and hex IC can be printed.

24.2 (lptpos) is not updated for the printing of characters less than hex 20, except for TAB (hex 09) and CR (hex 0D).

24.3 Except for extreme circumstances, no problem is caused. If the :

a. only characters printed are between hex 11 and IC,
b. printer head was on column one to start with,
c. LPRINT statements end with (;),
d. line is less than the page width of the printer,

-then the last line will stay in the buffer and not be printed when the program ENDs.
The buffer can be emptied be issuing a carriage return (:LPRINT:).

(also BASIC version 1.0)

25. JOYSTICK INTERFACE

25.1 The joystick interface is optional. Programs written that use joysticks should include the choice of using the keyboard if the joystick interface is not installed (see 25.6 below).

25.2 The joystick interface is available in two forms :

- low cost optional interface,
- incorporated in the VG 5216 extension.

25.3 If the low cost option is installed it can be assumed that at least one joystick is available.

25.4 If the VG 5216 extension is installed (which can be detected from the increase in RAM size) there may, or may not, be any joysticks available.

25.5 The presence of a joystick interface can be determined by ORing the results of scanning the 3 BIOS routines for both joysticks. If the interface is installed the result will consistently be zero. If it is not installed an indeterminate value will result. For this test, the joystick(s) must not be operated at the same time.

25.6 Programs should read the specified standard keys at start time :

Key "1"	:	One player/Joystick
Key "2"	:	Two players/Joystick
Key "3"	:	One player/Keyboard
Key "4"	:	Two players/Keyboard

The one player joystick answer assumes the "right" joystick only used. The two players keyboard answer assumes the standard keyboard definition for two players is used.

(also BASIC version 1.0)

26. MACHINE-CODE PROGRAMS ON CASSETTE

26.1 It is a truism that any program that is loaded from cassette is loaded in a BASIC environment, and that the first program loaded from cassette is loaded in a standard BASIC environment.

26.2 If a machine-code program can load and start execution in a standard BASIC environment, one line of BASIC is necessary to transfer control, e.g :

```
OCALL18951:< RET >
```

This line will transfer control to address hex 4A07.
The memory-map alter entering this line is shown as follows :

```
FORI:18940TO18953:?IPEEK(I):NEXTI < RET >
```

Note that 18951-18953 are zeros and can be overwritten by machine-code.

26.3 To save the program to cassette, enter :

```
CSAVEL:POKE18569,p:CSAVEM"filnam",18940,Lenth,0:POKE18569,0 < RET >
```

where :

p	= protection byte "allflg" (see TB 14 Piracy protection)
filnam	= name of program
Lenth	= length of program (counting from 18940)

Be careful to ensume that lenth doesn't extend into the BASIC stack -if uncertain, execute the save as described above and see if the end of the program is overwritten by stack entries.

26.4 If a non-standard environment is used for the program, it must be loaded in 2 or more, stages. The first stage loaded would automatically run to alter the environment for the next stage(s). This first stage would then load the subsequent stages (also see TB 17 Input streams. A cassette load command can be the last command in an ASCII file).
(also BASIC version 1.0)

27. PLAY COMMAND

27.1 If the PLAY command is entered via the BIOS "play" entry point at hex 008F, the stack is set up incorrectly. On exit from the routine the return address will be discarded prior to the<RET>.

27.2 To use the PLAY command, an extra stack level should be created before jumping to the "play" entry point :

```
                :
                CALL PLAYX          Play a string of notes
                :
     PLAYX      PUSH HL             Create dummy stack entry
                JP PLAY             Go to it
```

27.3 The exit from the PLAY command resets two screen associated variables :

"intdiv" to 3
"intact" to 1

If a screen update is not required, this can be prevented by temporarily changing index register IX prior to entering the PLAY command. "intdiv" is at address IX+3, and "intact" is at address IX+1. Obviously, the replacement value for index register IX+1 and +3 will be altered.

(also BASIC version 1.0)

28. CALL COMMAND

28.1 Register pair HL is used as the BASIC text pointer. The routine that is CALLed should preserve register pair HL.

28.2 If an index register is required, first use index register IY, which BASIC doesn't use.

If it is necessary to use register IX, the existing value should be preserved. Before the contents of IX are altered, interrupts should be disabled, and not reenabled until the original contents have been restored. Note also that some routines cannot be called after IX has been altered :

0018	outdo
001B	setext
008F	play
009E	cls
00A1	cll
00AA	kbscan
00B9	break

Screen updating via the resident interrupt handler should be prevented, and any entry point that returns control to BASIC should not be used.

(also BASIC version 1.0)

29. CONT COMMAND

29.1 If a program is restarted using the CONT command, two screen parameters are not correctly restored. The cursor remains visible, and continuation lines are enabled.

29.2 After CONTinuing, the two screen parameters will be restored following an INPUT command.

29.3 If, for debugging purposes, the two screen parameters must be restored, the following direct statement should be used :

```
POKE18429,PEEK(18429)AND191:POKE18545,PEEK(18545)AND64:CONT < RET >
```

(Also BASIC version 1.0)

30. CSAVE/CLOAD COMMANDS

30.1 Unless a full CLOAD command is executed, including the start line number option, it must be the last command on that line. this restriction applies to both immediate and deferred commands. This bug also prevents such a line having a comment (REM statement). The usual symptom from failure to observe this precaution is a "Missing line number" error in the BASIC line containing the CLOAD.

30.2 As well as CSAVEing and CLOADing BASIC text and machine code, three data structures can also be recorded on cassette. Two of the structures are simple :

CSAVE*	Numeric array
CSAVEX	String

Points to note when CLOADing such files are :

- Ensure the data name to receive the file has been referenced prior to the CLOAD (initialised),
- The CLOAD data name doesn't have to be the same as that used in the CSAVE,
- The CLOAD data file can be smaller (or the same size) as the data element to receive the file,
- Numeric arrays can be redimensioned by the CLOAD,
- If the string data is a BASIC program literal, the actual program line will be changed by the CLOAD.

30.3 The third data structure is string arrays :

CSAVE*	String array

This command only saves the pointers to the string elements. In order to CSAVE and CLOAD the string array data it must be in contignous memory locations, and written to cassette using CSAVEM, as a second file.

Initially, this may seem to be a difficult task, but with the exception of two PEEKS, can be easily accomplished in BASIC. An example of building some arrays and recovering them are shown on the next two pages.

Points to note when creating string arrays are :

- The CLEAR command reserves string space for the string array and computed string variables,
- A check is made to ensure we don't overflow our string array memory space,
- A PEEK is made to the BASIC variable (stktop) to find out where the string array space is located. Note the offset is always +1 the amount of computed string variable space,
- The CSAVEM always saves the whole array space, regardless of actual usuage,
- If the program loops back to line 110 for another file, any computed variables are lost by the CLEAR.

Points to note when reloading string arrays are :

- The CLEAR command is identical to that used to create the string arrays,
- Prefill the string array elements with maximum size computed variables. This will ensure that any other string variables that are computed will not be overwritten later,
- If the program processes more than one fill, the program restarts at line 180. It is not necessary to reinitialise memory or the array space.

Also note the example is designed for a basic VG 5000 console. For a larger memory the second parameter of the CLEAR command will have to be a negative value. Also, if variable S (in line 270 of creating string arrays) exceeds the value 32767, then 65536 should be substracted from it before use in the CSAVEM.

(Also see TB 12-Basic version 1.1, and VG 5000 Users Manual)

(also BASIC version 1.0)

Program to create string arrays on cassette
==

```
10 REM ***************************
20 REM *                         *
30 REM *   CREATE STRING ARRAY   *
40 REM *                         *
50 REM ***************************
60 :
70 :
80 REM The program always restarts here
90 REM in order to fix the array loc.
100 :
110 CLEAR 200,32764:REM Fix memory size
120 DIM X$(9):REM Array of 10 * 10 max.
130 :
140 REM Input the array data
150 L=0:FOR I=0 TO 9
160 PRINT"For ";I;:INPUT " enter word: ";X$(I)
170 L=L+LEN(X$(I)):NEXT I
180 IF L>100 THEN PRINT "Too many characters":GOTO 110
190 :
200 REM Name the file
210 INPUT "Enter file number: ";X
220 IF X<1 OR X>20 THEN PRINT"Sorry":GOTO 210
230 N$="File "+CHR$(X+48)
240 :
250 REM Save the file in two parts
260 CSAVE* N$ X$:REM Save pointers
270 S=PEEK(18581)+PEEK(18582)*256+101:REM Find last 100 bytes of strin
280 CSAVEM N$,S,100:REM Save strings
290 :
300 REM Are we finished ?
310 INPUT "Another file";Z$
320 IF Z$="N" THEN END
330 GOTO 110
```

Program to reload string arrays from cassette

```
10 REM ****************************
20 REM *                          *
30 REM *    RELOAD STRING ARRAY   *
40 REM *                          *
50 REM ****************************
60 :
70 :
80 REM The program always starts here
90 REM in order to reserve array space
100 :
110 CLEAR 200,32764:REM Fix memory size
120 DIM E$(9):REM Array of 10 * 10 max.
130 FOR I=0 TO 9:E$(I)="*"+"*********":NEXT I
140 INPUT "Enter your name: ";P$
150 :
160 REM Main loop of program
170 REM Reload array of your choice
180 INPUT "Enter file # to load: ";X
190 IF X<1 OR X>20 THEN PRINT"Sorry":GOTO 180
200 N$="File "+CHR$(X+48)
210 CLOAD* N$ E$:REM Load pointers
220 REM Load data
230 CLOAD
240 :
250 REM Main program goes here
260 FOR I=0 TO 9:PRINT E$(I):NEXT I
270 :
280 REM Are we finished ?
290 INPUT "Another file";Z$
300 IF Z$="N" THEN PRINT"Bye, ";P$:END
310 GOTO 180
```

31. RESET

31.1 RESET refers to the simultaneous pressing of the CTRL and <DELTA> keys. For previous information on RESET see TB 14-BASIC version 1.0.

31.2 To transfer control from RESET to a machine code program, the following procedure should be followed :

- at program initialisation, change the (nmihk) vector in addresses hex &"47EF" and hex & "47F0" to point to your own restart routine.

- The restart routine you code should start :

```
CALL &"0074"  Perform a RETN
DI            If required
JP   XXXX     Go to where you want
```

31.3 To be able to automatically re-run a BASIC text program from the beginning by pressing RESET, 12 bytes of machine-code must be included in the program, which will start with two POKES :

- at program initialisation, change the (nmihk) vector in addresses 18415 and 18416 to point to the machine code routine ; and also create this routine,

- In the routine, reset the BASIC interpreter to restart at the first line number in the program.

(also BASIC version 1.0)

Program to restart BASIC using CTRL+<DELTA>
==

```
10 REM ******************************
20 REM *                            *
30 REM *   RESET RESTART IN BASIC   *
40 REM *                            *
50 REM ******************************
60 :
70 :
80 REM An arbitrary memory address is
90 REM used to hold the 12 bytes of
100 REM machine code.  The address of
110 REM this routine is &"6000" (24576)
120 :
130 REM Initialization routine
140 RESTORE 250
150 FORI=0TO11:READ Z:POKE 24576+I,Z:NEXT I
160 POKE 18415,0
170 POKE 18416,96
180 :
190 REM Program goes here
200 PRINT"Hello and"
210 PRINT"Goodbye!"
220 GOTO 210
230 :
240 REM Data for initialization POKEs
250 DATA&"CD",&"74",0,&"FB",&"21",&"EA",&"24",&"E3",&"AF",&"C3",&"9A",&"25
260 :
270 :
280 REM The explanation of this coding:
290 :
300 REM     CD 74 00  Call RETN
310 REM     FB        Ensure interrupts
320 REM     21 EA 24  Point HL to newstt
330 REM     E3        Swop HL onto stack
340 REM     AF        Clear A and Z flag
350 REM     C3 9A 25  Jump to BASIC run
360 :
370 :
380 REM NB: Many rapid presses of CTRL+
390 REM     <DELTA> could possibly fill
400 REM     all the stack space and
410 REM     crash the program.
```

32. BASIC'S LINE FORMAT

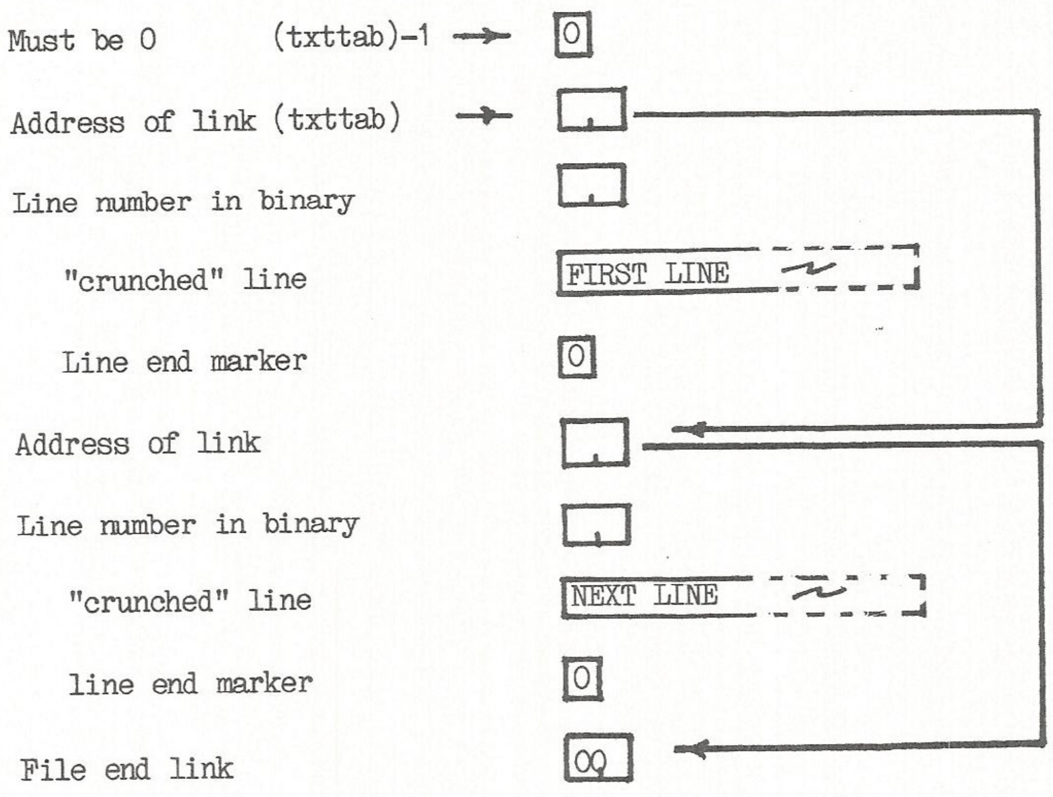
32.1 When a line of BASIC text is entered, it is "crunched" (tokenised) to take up less room, and speed execution.

32.2 The tokenised lines are pointed to by a BASIC variable (txttab) at address hex &"488E" and hex &"488F" (decimal 18574 and 18575). (txttab) always points to the first line in the program.

32.3 BASIC has a zero (nul) byte just before the program at (txttab)-1.

32.4 The program lines are chained together, the end of the chain (and hence the program) being two zero (nul) bytes. (txttab) always points to the first chain link.

32.5 The method of linking is as follows :



32.6 Zero (nul) bytes only appear in the program in two places :

- as a line end marker,
- as a value within a line number constant.

BASIC always ensures that 3 zero (nul) bytes apprear at the end of the program text. The symptom of losing the file end zero (nul) bytes is that the program may well run (if the highest line number is a GOTO or END, etc) but will go berserk at the end of a LIST command.

32.7 A line number constant is a line number in binary, created in command lines that contain line numbers, such as GOTO and GOSUB. BASIC recognises line number constants as they are immediately preceeded by a token of hex &"0E" (decimal 14).

32.8 Text relocation is easy. It can be seen that the only location specific information stored in the program are the link addresses. The first byte relocated must be the initial zero (nul) byte. (txttab) is then reset to point to first link address (the next byte) and routine "fini" is jumped to to calculate the new link addresses (see TB 15-BASIC version 1.1). Should control need to be kept and/or variables not cleared, a call to "rchain" at address hex &"1C23" (decimal 7203) will relink the program. (Also see TB 33).

(also BASIC version 1.0)

33. BASIC ENTRY POINTS

33.1 This Bulletin continues on from TB 15-BASIC version 1.1.

Name Address	Description parameter

33.2 Two useful addresses :

jingle 116D	Sign-on jingle string. Point HL to jingle and call "playit" routine (see below).
reddy 2214	BASIC's Ok! string. Point HL to 2214 and call "strout" routine (see below).

33.3 Some BASIC subroutines that can be CALLed (except for "run") :

playit
0A75

Will play an ASCII string enclosed in quotes that conforms to BASIC's PLAY command format. Control is passed via the (plyhk).

entry	HL = address of first quotation mark
exit	errors in the string return control to BASIC with an error message
modifies	all

rchain
1C23

Rechains the link addresses in a BASIC program.

entry	none, but (txttab) must be good
exit	HL = value for (vartab)
modifies	HL, DE, A

head
1F9C

Checks that there is at least 80 stack levels between the current stack position and the value in HL.

entry HL = address to be checked for validity
exit Cflag off = no room
Cflag set = room
modifies A only

run
259A

Vector of the RUN command in BASIC. No line number parameter can be given. Because of (SP) can only be JUMPed to.

entry A = zero
Z flag set (i.e : zero)
(Sp) = address of "newstt" routine (&"24EA")
exit none
modifies all

clearc
2EE8

Reinitialises a BASIC program by doing a CLEAR, RESTORE and resetting the stack.

entry none
exit HL is preserved
modifies all escept HL. The stack is now empty.

getcar
32B7

Performs a full keyboard scan as used by BASIC's text editor. The character is not echoed to the screen (or printer). All features are supported. The routine doesn't wait for a key press.

entry none
exit Value is in A and (crchar).
0 = no key pressed
cflag set = repeated key
cflag off = new key
2flag set = no key
modifies A only

strout 36AA	Will print a string of up to 255 characters on the current output device. HL should be set to point to the start of the string. The string is in ASCII and terminates with a zero (nul) byte. entry HL = address of first byte in string exit none modifies all
crdonz 3C40	Sends a "visual" character (hex &"OF"- decimal 15) to the screen, and then performs a cr/If if the cursor is not pointing to the first column of a screen line. entry none exit none modifies A
crdonx 3C43	As for "crdonz" above, but omits the "visual" character output.
inlin 3CA2	Enters the BASIC text editor. When RET is pressed, the current line the cursor was in is returned in (buf). This overwrites BASIC's buffer contents. (buf) starts at hex &"4898" (decimal 18584) entry none exit HL points to (buf)-1 The line end is marked with a byte of zero (nul) Cflag off = line returned Cflag set = SHIFT/STOP pressed modifies HL, DE, A

34. MEMORY MAP

34.1 The memory map for a basic VG 5000 console appears on page 86 (Annexe 2) of the VG 5000 User Manual.

34.2 If a VG 5216 extension is fitted, available RAM is increased by 16 K. BASIC will find and use this additional RAM. the usable memory is increased by 16 K, and the stack and computed string variable space are moved to the new top of RAM area. The sign-on message now displays "30142 bytes free".
Two of BASICs variables are reset :

(stktop) will increase from hex &"7FCA" to hex &"BFCA"

(memsiz) will increase from hex & "7FFC" to hex &"BFFC"

34.3 If a 16 K RAM cartridge is fitted in the VG 5216 extension, another 16 K of memory will be found and used by BASIC. Again, the stack and computed string variable space are moved to the new top of RAM area. The sign-on message now displays "46526 bytes free".

Two of BASIC's variables are reset :
(stktop) will increase to hex &"FFCA"
(memsiz) will increase to hex &"FFFC"

35. PIRACY PROTECTION

35.1 Bulletin 14 (BASIC version 1.1) gives the essential information to prevent unauthorised access to programs that are written in BASIC or machine code. A further explanation of the example in TB 14.6 on page 2/2 is given below.

35.2 A machine-code program is difficult to "break" into as it will always load at its CSAVE address upwards (also see Bulletin 26). if this address is the usual one that is held in (txttab)-1, hex &"49FB" (decimal 18939), and "allflg" is set by a POKE prior to the CSAVEM, no further protection is usually needed. It is , of course, possible for the program to check "allflg" prior to running to make sure it has the correct value expected, which will be set as previously described.

35.3 A BASIC text program is easier to "break" into, as it can be appended to a program that is designed to CLOAD it and then sets the "allflg" to zero and ENDS. Even if the protected program initially jumps to a strange line number to commence, this can be "locked out" by the breaker program. What the routine in TB 14.4 does, as well as checking that "allflg" is correct, is that it also checks some location-dependent information known only by the games writer by PEEKing it. If the values are wrong, either the program is loaded at a new address, or has been altered. In either case, we restart the computer from cold. Such checks could be scattered about the program to make them difficult to find.

36. CLEAR COMMAND

36.1 Care should be taken before using a CLEAR Command with a second parameter, as it will alter BASIC's "memsiz" variable. For previous information on CLEAR see TB's 10 and 21 - BASIC version 1.1.

36.2 "memsiz" holds the upper limit of memory used by BASIC. Any memory above "memsiz" is protected from BASIC (except for the POKE command). Machine-code subroutines that are to be published on cassette will be located above "memsiz" and linked into BASIC using the hook vectors. Some subroutines are fully relocatable. No permanent space is allocated for possible disk workspace. This workspace and file buffers will again be located above "memsiz".

36.3 The current "memsize" can be obtained by :

```
1000    A = PEEK (& "491F") + (& "4920") * 256
```

If the programmer wishes to use workspace protected by "memsiz" this old value should be used as the upper limit of memory availability. If this advice is ignored, the programmer may overwrite a BASIC subroutine and crash the computer.

36.4 If "memsiz" is examined and is not the correct value to permit the program to run, a graceful exit is to jump to, or call & " 2ED2" for "Out of memory" error.
The usual values for "memsize" are :

16K ram :	32764
32K ram :	49148
48K ram :	65537

(also BASIC version 1.0)

37. BASIC'S HOOKS

37.1 In general, programs can be divided into two main categories :

Closed end
Open end

The "Closed end" programs never return to BASIC, and will probably never have to coexist with BASIC subroutines. They will normally set the hooks how they require to operate and change registers and values in memory as required. Most games are in this category.

The "open end" programs return to BASIC, either because the user has finished with the program, or to load another program. Game initial screen programs and some commercial programs are in this category.

37.2 "Open end" programs should take case when altering hooks, registers and values in memory, as the return to BASIC may sometimes fail (eg : CLOAD). The main points to observe are :

- Restore the initial values of HL and IX.
- Restore, or include on a chain, the hooks used.

In particular, avoid :

- Disconnecting "inthk" (it should be a jump or return ; do not alter to INC SP/INC SP/EI as this is no longer a hook but coding dependent on the adjacent hook "calhk".
- Altering IX with interrupts enabled.
- Changing the stack pointer
- Overwriting BASIC'S workspace.

(Also BASIC 1.0)

38. FOR...NEXT COMMAND

38.1 The next statement always transfers BASIC's program flow to the associated FOR command. Whilst the FOR statement is true, the program lines that follow it will be executed. On the first occasion that the FOR statement is false, control is passed to the program lines following the NEXT.

38.2 A list of all true FOR statements is kept on the stack ; the entries are made according to the order in which they were executed (which is not necessarily the same as the order in the program listing).

38.3 When a FOR statement becomes false, it is removed from the list. All lower (i.e : subsequent) FOR statements are also removed, if present.

38.4 If a FOR statement is executed again that has previously become false,it will not be on the list and will be rebuilt on the current end of the list. If the FOR statement is currently true, it should be rebuilt where it is without disturbing the higher list entries. However, there is a bug, and a FOR statement that is currently true destroys the whole list, causing a "Syntax error" in the line number that contains it.

38.5 There are three ways to avoid the problem :

- always execute a NEXT until the FOR becomes false.
- before reexecuting the FOR, reset the associated variable to its TO loop value and execute a NEXT. This will force the FOR to be false, and be removed from the list. Any FOR statements higher on the list will be preserved. GOSUB context is preserved.
- Reset the associated variable to its initial value and reenter the loop omitting the FOR command. Any FOR statements higher on the list will be preserved. GOSUB context is destroyed, so this method is not recommended.

(also BASIC version 1.0)

```
 10 REM ************************************
 20 REM *                                  *
 30 REM *   CORRECTING FOR...NEXT CONTEXT    *
 40 REM *                                  *
 50 REM ************************************
 60 :
 70 :
 80 REM EXAMPLE 1
 90 :
100 REM Force the NEXT to make the FOR statement false
110 :
120 FOR A=1TO20
130 PRINT "loop 1" ; A
140 IF A=11 THEN 160
150 NEXTA
160 FOR B=10TO5 STEP-1
170 PRINT "loop 2" ; B
180 IF B=6 THEN 200
190 NEXT B
200 PRINT "Redo" : A=20 : NEXT A : GOTO 120
500 :
510 :
520 REM EXAMPLE 2
530 :
540 REM Reset the FOR variable without executing the FOR statement
550 :
560 FOR A=1TO20
570 PRINT "loop 1" ; A
580 IF A=11 THEN 600
590 NEXTA
600 FOR B=10TO5 STEP-1
610 PRINT "loop 2" ; B
620 IF B=6 THEN 640
630 NEXT B
640 PRINT "Redo" : A=1 : GOTO 570
```

Example :

```
Loop 1 1
Loop 1 2
Loop 1 3
Loop 1 4
Loop 1 5
Loop 1 6
Loop 1 7
Loop 1 8
Loop 1 9
Loop 1 10
Loop 1 11
Loop 2 10
Loop 2 9
Loop 2 8
Loop 2 7
Loop 2 6
Redo
Loop 1 1
Loop 1 2
Loop 1 3
Loop 1 4
Loop 1 5
Loop 1 6
Loop 1 7
Loop 1 8
Loop 1 9
Loop 1 10
Loop 1 11
Loop 2 10
Loop 2 9
Loop 2 8
Loop 2 7
Loop 2 6
Redo
Loop 1 1
Loop 1 2
Loop 1 3
Loop 1 4
Loop 1 5
Loop 1 6
Loop 1 7
Loop 1 8
Loop 1 9
Loop 1 10
Loop 1 11
Loop 2 10
Loop 2 9
Loop 2 8
Loop 2 7
Loop 2 6
Redo
Loop 1 1
Loop 1 2
Loop 1 3
Arrêt en 130
Ok!
```

39. CALL COMMAND

39.1 Further to TB 28-BASIC version 1.1, it should be noted that the CALL hook vector at address &"47D3" (18387) can only be set up by the CALL command.

39.2 The CALL hook vector "calhk" is modified by the CALL command to contain :

```
        C3              JUMP -
        nnnn            - To CALL address
```

It is not possible for a user to revector a CALL by changing "calhk" as the contents will be overwritten in every case.

(also BASIC version 1.0)

40. STORE/SCREEN/DISPLAY COMMANDS

40.1 These three commands control the video screen refresh rate (i.e : when the internal ram screen area is transfered to the video controller for displaying).

40.2 The measurement of elapsed time is made in 20 msec units, which correspond to the vertical blanking (VS) period of the video controller. For each VS, a maskable interrupt is generated, which (if interrupts are enabled) suspends the current program and transfers control to address hex 0038.

40.3 For transfer to take place, four conditions must be satisfied :

- interrupts enabled
- refresh counter decremented to one
- screen update flag set
- "inthk" returns control to the screen transfer routine.

The refresh counter "intdiv" is at location IX, the screen update flag "intact" is at location IX +1 and the preset for the refresh counter "intrat" is at location IX +2.

40.4 If interrupts are disabled, all counter are suspended and no video screen refresh can occur. If the disable lasts for $>\approx$ 108 msec, one or more interrupts may be lost, disturbing the counting. BASIC commands that can do this are CSAVE/CLOAD/SAVE/LOAD/PLAY and disk I/O.

40.5 For further information on interrupts, see TB 0041.

40.6 STORE command

40.6.1 The STORE command sets "intdiv" and "intrat" to zero. This freezes the video screen display for the longest interval possible. The screen is not refreshed prior to the delay. The current program continues to update the internal ram screen area.

40.6.2 The following commands will cause an immediate transfer after a STORE has been executed :

- scrolling of the screen
- homing the cursor
- clearing the screen
- SCREEN

Note that the PAGE command will disable scrolling.

40.6.3 The following commands also cause an immediate transfer after a STORE has been executed, and also change the refresh rate :

- DISPLAY
- INIT
- END
- STOP
- error condition

For every case but DISPLAY, the "intrat" is restored to the standard value of 10 (hex 0A).

40.7 SCREEN command

40.7.1 The SCREEN command sets "intact" and "intdiv" to 1. This causes an immediate transfer on the next interrupt, even if the internal screen area has not been altered.

40.7.2 The screen refresh rate will be unaltered.

40.7.3 Note that in all instances of immediate screen tranfer, the next refresh has been "brought forward". The immediate tranfer is not performed as well as the regular refresh.

40.8 DISPLAY command

40.8.1 The DISPLAY command sets "intact" and "intdiv" to 1. This will cause an immediate transfer on the next interrupt, even if the internal screen area has not been altered.

40.8.2 The screen refresh rate is altered, either to the value given, or a default value of 32 (hex 20).

40.8.3 Due to a bug, the counter runs four times as fast as it was intended to. The actual values for the DISPLAY command change the refresh rate as follows :

1-4	refresh in	20 msec
5-8	"	40 msec
9-12	"	60 msec
13-16	"	80 msec
29-32	"	160 msec (default)
245-248	"	1.24 sec
249-252	"	1.26 sec
253-0 (256)	"	1.28 sec

Note that for the automatic refresh to occur all the conditions in 40.3 above must be true. The actual refresh takes approximately :

60 msec BASIC version 1.0
40 msec BASIC version 1.1

BASIC version 1.0 loses 3 interrupts ;
BASIC version 1.1 loses only 1 interrupt.

40.9 For an example of a program using these commands, see the next page.

(also BASIC version 1.0)

Demonstration of STORE - SCREEN - DISPLAY commands

```
10 REM  _______________________
20 REM |                       |
30 REM | INSTANT SCREEN CHANGING |
40 REM |_______________________|
50 :
60 :
70 GOTO 130:REM Jump to start
80 :
[illegible] REM "Generator" noise subroutine
100 FORK=1TON:FORD=0TO10:SOUND244,1,D:NEXTD:FORD=11TO1STEP-1:SOUND244,1,D:NEXTD
NEXTK:RETURN
110 :
120 REM Set up initial values
130 M$="RANDOM BAR CHART GENERATOR "
140 X$=CHR$(127):Y$=X$+X$+" ":L=&"47FD":INIT0,0:N=10:GOSUB 100
150 :
160 REM Main loop
170 REM - set up and print columns
180 STORE:FORH=1TO37STEP3:TXINT(RND(4)*7+1):FORV=INT(RND(2)*23)TO23
190 CURSORXH:CURSORYV:PRINTY$;:NEXTV,H:TX7
200 :
210 REM - turn on cursor & print title
220 SCREEN:POKEL,64:FOR Z=1TOLEN(M$):PRINTMID$(M$,Z,1);:DISPLAY:SOUND251,1:NEXT
:N=3:STORE:GOSUB 100
230 :
240 REM - clear the "hidden" screen
250 FORV=0TO24:CURSORX1:CURSORYV:PRINTCHR$(4);:NEXTV:POKEL,0:GOTO 180
```

41. INTERRUPTS

41.1 Two type of interrupts are supported :
- non-maskable
- maskable (mode 1)

41.2 Non-maskable interrupts (NMI) are used for two purposes :
- distinguishing between French/Export models
- support of RESET feature

The RESET feature was discussed in BASIC version 1.0 - Technical Bulletin 14.

41.3 Maskable interrupts are enabled in mode 1, which vectors control via address hex 0038 (RST 7). The primary function is to synchronise internal ram screen area to video controller transfer, with the vertical blanking (VS) period.

41.4 When a maskable interrupt accurs, control passes to the "inthk" at address hex &"47D0". This normally contains a RET instruction to return control to the screen transfer routine.

41.5 If screen transfer takes place, it uses approximately :

61.053 msec	BASIC version 1.0
39.972 msec	BASIC version 1.1

As a result, transfer will mask 3 interrupt times for BASIC version 1.0, and only 1 for BASIC 1.1.

The reason for this is that the interrupt line is active for the length of time that VS occurs. This is :

124 µsec	- non interlacing
160 µsec	- interlacing

41.6 If a user program utilises "inthk" but still uses BASIC's screen transfer routine, a check should be made to ensure that VS is still active before returning to that routine, otherwise the video screen may flicker or be temporarily corrupted.

41.7 Other interrupt causing devices may be connected to the VG5000. In these cases it is important to distinguish which device caused the interrupt.

41.8 Small routines that use VS for synchronisation (e.g : a 20 msec timer routine) must ensure that VS has ended before enabling interrupts again, otherwise the routine may run several times in the same interrupt period. BASIC itself has a bug of this nature that affects the counting associated with the DISPLAY command (see TB40).

41.9 Games programs that use VS for synchronisation would usually have relatively lengthy tasks connected to "inthk". However, if there is a possibility of an "empty" task (e.g : a very low skill level for a game), an idle task must be used to maintain synchronisation.

41.10 In order to meet all the above requirements, a simple piece of machine-code is needed that will examine the VS status bit in the video controller (vgp). The mask for this bit is set to follow the state of VS :

```
        3E 20          LD             A,H'20'
        D3 8F          OUT            (H'8F'),A
        DB CF          IN             A,(H'CF')
        CB 57          BIT            2,A
```

If VS is still active, the Z flag is set. If it is required to wait until VS has finished, the next line of code would be :

```
        28 FA          JR             Z,H'FA'
```

(also BASIC version 1.0)

42. BASIC'S CURRENT LINE POSITION

42.1 As a result of calculating which line number is to be executed next, BASIC leaves a pointer to that line in a variable.

42.2 It is possible to recover the address in memory of the current line, if the following statement is placed at the start of that line :

```
V=PEEK(18895)+PEEK(18896)*256
```

V must follow the rules for memory addressing in BASIC, so before use, an adjustment should be made :

```
IF V > 32767 THEN V=V-65536
```

42.3 An example of the use of this technique would be to dynamically change the contents of a SETEG or SETET statement :

```
990     V=18895
1000    SETEG m, "AAAAAAAAAAAAAAAAAAAA"
1010    V=PEEK(V)+PEEK(V+1)*256-22
1020    IF V > 32767 THEN V=V-65536
```

Note the adjustment of -22 to V. V now points to the " at the start of the SETEG parameter. Obviously, no further statement is allowed between the SETEG and the calculation of V!

(also BASIC version 1.0)