0. INDEX

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 1.0 - if a further version is released for production a new series of Bulletins will be issued.

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.

TO BE CORRECTED

2. HEXADECIMAL NUMBERS

2.1 Numbers can be expressed in hexadecimal notation by &"XXXX" where XXXX is 0-4 hex. digits. In general, integers used in functions have the same restrictions as decimal numbers and range from -&"8000" to &"7FFF". Where they are used in an expression as a constant they have the range -&"FFFF" to &"FFFF".

2.2 Hexadecimal numbers are not correctly set up in the floating-point accumulator. The symptoms are wide ranging, but usually result in unpredictable execution of BASIC statements that contain them. Sometimes they will work correctly, sometimes not.

2.3 If hexadecimal notation is required, and unreliable results are obtained, the solution is to force the floating-point accumulator to a correct format prior to evaluating the hexadecimal number. The simplest way is to set a dummy variable to zero.
For example :

```
100  A = 0 : FOR LOOP = -&"7000" TO -&"6FF0"
110  PRINT LOOP
120  NEXT LOOP
```

- Will always run correctly. To see the bug, remove the "A = 0".

2.4 If a hexadecimal number is followed by one or more traling spaces, a "syntax error in XXXX" is incorrectly given.

TBC

3. SCREEN EDITING

3.1 The BASIC Screen Editor is a powerful, easy-to-use tool for entering and changing programs. However, its flexibility also means that there are a number of extreme circumstances that the user can get into where the result will not be entirely as expected. These circumstances are still being investigated, but two instances are given here.

3.2 On a multiple program line, if the DELETE key is used to erase some characters, when it crosses the boundary between column 1 of the line and column 39 of the previous line if jumps one character extra to the left, deleting character 38 and preserving character 39.

3.3 When entering lines using the AUTO command, a line that is exactly filled, so that the cursor is in column 1 of the next line, will exit the AUTO mode when RET is pressed.

4. AUTO COMMAND

4.1 Whilst entering program lines, it is possible to drop out of the AUTO mode. See TB 3.3.

4.2 One issued, the AUTO command continues to operate even whilst direct commands are executed. This is quite useful ; eg. LIST, but the BASIC fails to kill the AUTO command for RUN and NEW which is undesirable.

TBC

5. LLIST/LPRINT COMMANDS

5.1 If the default (rawprt) and (prtxlt) values are used to produce VG 5000 characters on an MSX printer (VW 0010 or VW 0020), the MSX character corresponding to hex F1, decimal 241 is printed as character hex F2, decimal 242.

5.2 This is due to an error in the translation table pointed to by (prtxlt) at offset hex F1. The symptom is ≥ obtained instead of ±.

5.3 To avoid the problem, if this character is required, turn off translation by temporarily setting (rawprt) to hex 01. Reset to hex 00 afterwards. (rawprt) is located at hex 4873, decimal 18547 ; and (prtxlt) is located at hex 4878/9, decimal 18552/3. For further information see the manual for the VG 5216 Extension.

6. VERSION NUMBER

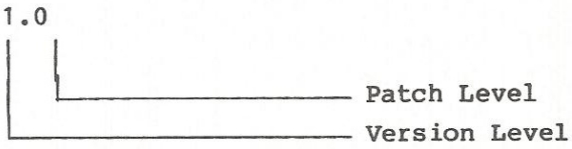
6.1 For purposes of upward compatibility with future hardware, it is important that the software always uses BIOS features in the BASIC for I/O processing.

6.2 Sometimes, because of bugs, or timing problmes, this is not possible to maintain. In such circumstances it is strongly recommended that the software should ensure that it is compatible with the users system by checking the Version No.

6.3 The following small routine demonstrates, in BASIC, where and what the Version No is:

```
?"Version ";CHR$(PEEK(4));CHR$(PEEK(5));CHR$(PEEK(6))
Version 1.0
Ok!
```

6.4 The number is composed as follows :



Patch Level will be changed as bugs are corrected or features are added. Software is upward compatible.

Version Level will be changed if a major structural change is made. Software would not be upward compatible.

7. JOYSTICK (2 ACTION BUTTONS)

7.1 The joystick interfaces support two action buttons. The BASIC only supports one, and therefore the BIOS entry points also only provide for one action button.

7.2 It is possible to access the I/O ports for the joystick to directly read the status of the second action button. This also enables multiple readout of the appropriate bits for multiple action (eg.: North/West/Action 1/Action 2) or for other hardware (eg. : paddles).

7.3 The code is as follows :

DB 07 Places left hand joystick value in register A

DB 08 Places right hand joystick value in register A

7.4 The value returned in register A is as follows :

Bit :	0 = 0	North / Forward
	1 = 0	East / Right
	2 = 0	South / Back
	3 = 0	West / Left
	4 = 0	Action 1
	5 = 0	Action 2
	6 = 1	Unused
	7 = 1	Unused
	0 = pressed	1 = not pressed

8. BASIC TEXT RELOCATION

8.1 BASIC text can be placed at a start address other than hex 49FC, decimal 18940. The required value for the pointer is held in (txttab) at addresses hex 488E/F, decimal 18574/5.

8.2 If (txttab) is changed, a NEW, CLOAD of text or SAVE of ASCII text will also change the following variables :

(vartab)
(temp)
(arytab)
(strend)

- so that they address free memory above the program text. CLOADA doesn't change the variables.

8.3 The two usual circumstances for changing (txttab) are :

- to create a "buffer" space starting at hex 49FB, decimal 18939.
- to point to tokenised text in a ROM.

8.4 Remember that when (txttals) is altered, (txttab 1) must be hex 00.

8. BASIC TEXT RELOCATION

8.1 BASIC text can be placed at a start address other than hex 49FC, decimal 18940. The required value for the pointer is held in (txttab) at addresses hex 488E/F, decimal 18574/5.

8.2 If (txttab) is changed, a NEW, CLOAD of text or SAVE of ASCII text will also change the following variables :

(vartab)
(temp)
(arytab)
(strend)

- so that they address free memory above the program text. CLOADA doesn't change the variables.

8.3 The two usual circumstances for changing (txttab) are :

- to create a "buffer" space starting at hex 49FB, decimal 18939.
- to point to tokenised text in a ROM.

8.4 Remember that when (txttab) is altered, (txttab-1) must be hex 00.

9. INPUT COMMAND

9.1 Multiple-variable input statements should be avoided. Unless the user enters all the variables on one line, further entries (with two "?" marks) are processed incorrectly.

9.2 If the second or subsequent variable is a string, it will have "??" prefixed to it.

9.3 If the second or subsequent variable is a number, it will be rejected as invalid, and all the variables will need reentering.

9.4 Although PAGE stops the screen scrolling for PRINT commands, it is not effective for INPUT commands.

10. PAGE COMMAND

10.1 The PAGE command doesn't stop the screen scrolling for INPUT commands -see TB 9.4.

10.2 The PAGE command is turned off incorrectly by an INPUT command. If the user input causes an error condition, of he goes off the end of a line, a scroll might occur when the RET key is depressed, spoiling the screen lay out.

11. SCREEN CONTROL CHARACTERS

11.1 The data characters and control characters for each of the four BASIC screen character sets are as follows :

	Data	Control
TX-	hex 10 - hex 1D hex 20 - hex 7F	hex 00 - hex 0F hex 1E & hex 1F
GR-	hex 00 - hex 7F	hex 80 - hex 8F hex 9E & hex 9F
ET- EG-	hex 20 - hex 7F	hex 00 - hex 0F hex 1E & hex 1F

There are particular points to note for each set.

11.2 TX- character hex 10 (..) is a specially preprogrammed character. It occupies the code hex 7F in the ET character set. If the ET character set is reprogrammed, (..) may disappear, and be replaced with a new character. (..) is only programmed at power on and RESET.

11.3 GR- The control codes have bit 7 set. All character codes between hex 00 and hex 7F print a character. All control codes with bit 7 set that "clear" the screen set the data value of that screen location to hex 00, rather than hex 20 as for all other sets.

Screen PRINT functions that work with the TX set have different actions in GR:

```
PRINT A$
```

- will terminate with the equivalent of CHR$(3);

```
PRINT A$,B$
```

- will infill between A$ and B$ with the equivalent of a string of CHR$(32)'s

```
PRINT SPC(I);
```

- will print the equivalent of CHR$(32); I times

11.4 ET- note that character hex 7F is already programmed -see 11.2 above. Hex 10 is translated to hex 7F. Hex 11 - Hex 1D are also translated, but to null printing values that appear as ( ). Hex 32 must be programmed to (space) for PRINT A$,B$ and PRINT SPC(I); to work normally.

11.5 EG- Hex 10 is translated to hex 7F. Hex 11 - hex 1D are also translated, but to null printing values that appear as ( ). Mixing TX and EG on the same screen line may result in different TX results, depending on which set prints first on the line. Hex 32 must be programmed to (space) for PRINT A$,B$ and PRINT SPC(I); to work normally.

12. VARPTR(X$) FUNCTION

12.1 This useful function is not provided. However it can be emulated.

12.2 In order to obtain the address of a string, first reference the string, then PEEK the associated pointer value. For example :

```
A = LEN (X$)
A = PEEK(18725)+ PEEK(18726)*256
```

A now contains the address of the first byte of X$.

12.3 If the string is empty, the length will be zero and the associated pointer will not be set.

13. SCREEN EDITING

13.1 If a program line is too long for the BASIC text buffer, when RET is pressed the cursor will stay where it is. The line is truncated and incorporated in the program. No other warning is given. Manuel-

13.2 On lines that are very close to the line length limit of 126 bytes, it may be possible to enter the line, LIST it, but be unable to completely re-enter it by pressing RET. For example, a line with a ? in it will be expanded by LIST to PRINT, thus increasing the line length. Manuel.

13.3 Editing a long program line starting in line 0 of the screen is not recommended. Inserting characters into screen line 0 will cause the statement to "roll-off" to the right, losing the line end characters. Inserting characters into other screen lines will scroll the first line of the statement off the screen. Note that this condition can be created during editing even if the program line didn't originally start in line 0.

13.4 If a program line is exactly one screen line long (or an exact multiple), altering the last (39th) character may link the next screen line to it as a continuation line -remember to key "DELETE TO END OF LINE" if this happens.

(also see TB 3)

4. RESET

14.1 The RESET is "soft", that is, under software control.

14.2 When RESET is pressed, an NMI occurs and control passes to the (nmihk) at address hex &"47EE" (decimal 18414). This normally contains a jump instruction that points to the "warmst" routine. This routine checks if CTRL is also pressed. If not, processing continues from where it stopped. If CTRL is also pressed control is passed to the "warmgo" routine. (This routine is also pointed to by the BIOS entry point hex &"33" (decimal 51).

14.3 The "warmgo" routine does the following :

- sets up a temporary stack pointer
- resets cassette and sound latches
- resets input stream to keyboard (console)
- resets printer strobe
- enables interrupts
- sets cursor to screen line 0, column 1
- resets to correct cursor shape and enables it
- restores the video mode and colours to the default
- resets the BASIC stack pointer
- plays the sign-on jingle
- reenters BASIC at the "ready" point.

14.4 If BASIC's system variables have be damaged in the meanwhile, RESET will not work properly.

14.5 As RESET will interrupt anything that is happening (NMI), any results stored in variables, cassette output, etc. may be invalid. The program cannot be "CONT".

14.6 The user can alter the (nmihk) bytes hex &"47EF" and &"47FO" (decimal 18415 and 18416) to point to another place for control over RESET.

14.7 To disable RESET, poke hex &"74" (decimal 116) into location hex &"47EF" (decimal 18415) and 0 into location hex &"47FO" (decimal 18416). If a renable is required, save and restore the original values of these two bytes.

TBC.

15. LOAD COMMAND

15.1 ALOAD command with no parameters doesn't always assume a start line number of zero.

15.2 For reliable LOADing of ASCII text files, always specify either the first line number required to be loaded, or a (,) for the whole file. For example :

```
LOAD1000      < RET >
LOAD,         < RET >
```

TBC.

16. POKE COMMAND

16.1 The second parameter (value to be POKEd) may be incorrectly evaluated.

16.2 Positive values of 0 to 255 are correctly evaluated. If the value exceeds 255, incorrect evaluation occurs, but no "Illegal Function call" message is given. The actual value POKEd will be modulo 256.

17.

CLEAR To be documented.

17. AUTO COMMAND

17.1 If the AUTO command creates a line number that already exists in the program, no warming is given.

17.2 The AUTO command "wraps-round" for line numbers over 65529 and creates incorrect line numbers from that point onwards.

(also see TB 4)

18. SCREEN EDITING

18.1 The "insert line" key is not disabled during an INPUT statement. Use of this key will cause a partial screen scroll, destroying the screen format.

18.2 If the cursor is moved to column 1 whilst in an INPUT statement, pressing RET will cause a partial screen scroll ("insert line"), destroying the screen format.

18.3 Use of the "insert character" key in an INPUT statement will eventually cause data to be shifted right out of column 39. When this happens an "insert line" is performed, destroying the screen format.

(also see TB 3 and TB 13)

19. SCREEN EDITING

19.1 If the < LIST > key is used to display BASIC program text, the last line displayed on the screen may have non-displayed continuation lines.

19.2 It is not recommended to edit the last line on the screen as pressing < RET > would shorten that line to its visible portion only.

(also see TB 3, TB 13 and TB 18)