VG5000 TECHNICAL MANUAL

S. Haigh 23 MAY 1984

INTRODUCTION

This manual is intended for those wishing to develop machine language applications software for the VG5000 home computer. The manual describes detailed information on the hardware organization, memory mapping, BASIC operation and useful BIOS features.

1 HARDWARE ORGANIZATION

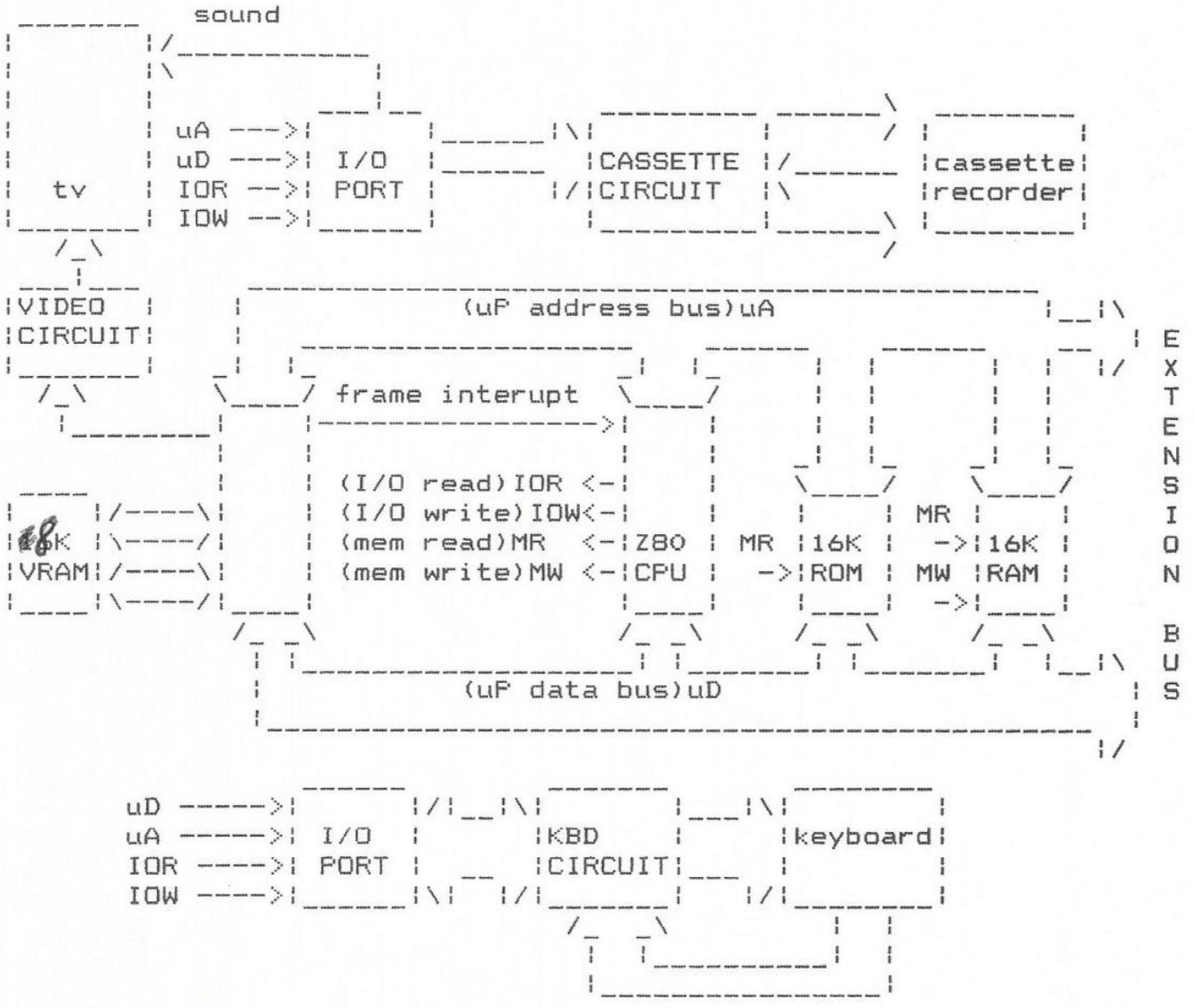


Fig 1.1 hardware block diagram.

Hardware organisation

1.1 General Description

The VG5000 home computer is based around a Z80A microprocessor and the Thomson-Efcis EF9345 video display processor. Built into the standard console is 16K ROM for onboard software, 16K RAM for processor work space and BASIC storage, cassette interface, keyboard and video circuits. An extension bus is provided which includes all major microprocessor signals and is used for the addition of extension devices such as hand control interface, expansion memory unit, disk, modem and light pen interfaces etc.

1.2 Microprocessor Circuit

The Z80A CPU is fed with a 4 MHz clock supplied by the video chip. There is a wait state applied to the processor on M1 memory fetches. This wait is not required for the onboard memory but is used as a precaution in case of slow memory devices being connected to the extension bus. The average processor speed is approximately 3.7 MHz with M1 waits. ~~The request is not used by any onboard circuitry, although it is available on the extension bus.~~ Data to the video chip is uploaded via I/O writes into the video system's own memory and common memory sharing which usually results in a large reduction of processor speed is not employed.

1.3 Video Circuit

The video system is based around the Thomsom-Efcis EF9345 single chip colour CRT controller. This chip is supplied with a 12MHz clock which it uses for its own dot clock and subdivides for a 4MHz microprocessor clock. The video chip has a number of internal registers for storing data such as cursor position, border colour etc and a 8K RAM store which it uses for the display image and programmable character shapes.

The microprocessor communicates with the video chip via I/O reads and writes to 2 port addresses &"8F" and &"CF". Additional information is supplied to the microprocessor by the frame blank start signal which is connected to the Z80 interrupt pin which enables the software to synchronise communication.

Red, green, blue, line blank and frame blank signals are provided to the video circuitry to produce television compatable video signals.

1.4 Keyboard Circuit

The keyboard keys are connected in a matrix format as seen in the VG5000 Manual. The keyboard is scanned by a sequence of I/O accesses to addresses &"80" through &"87" (excluding the reset key which is hard-wired). Data returning from the matrix is interpreted by the microprocessor to determine the location of any active keys. Key debouncing is performed by software.

1.5 Cassette Circuit

See attached documentation for VG5000 cassette.

2 MEMORY MAP

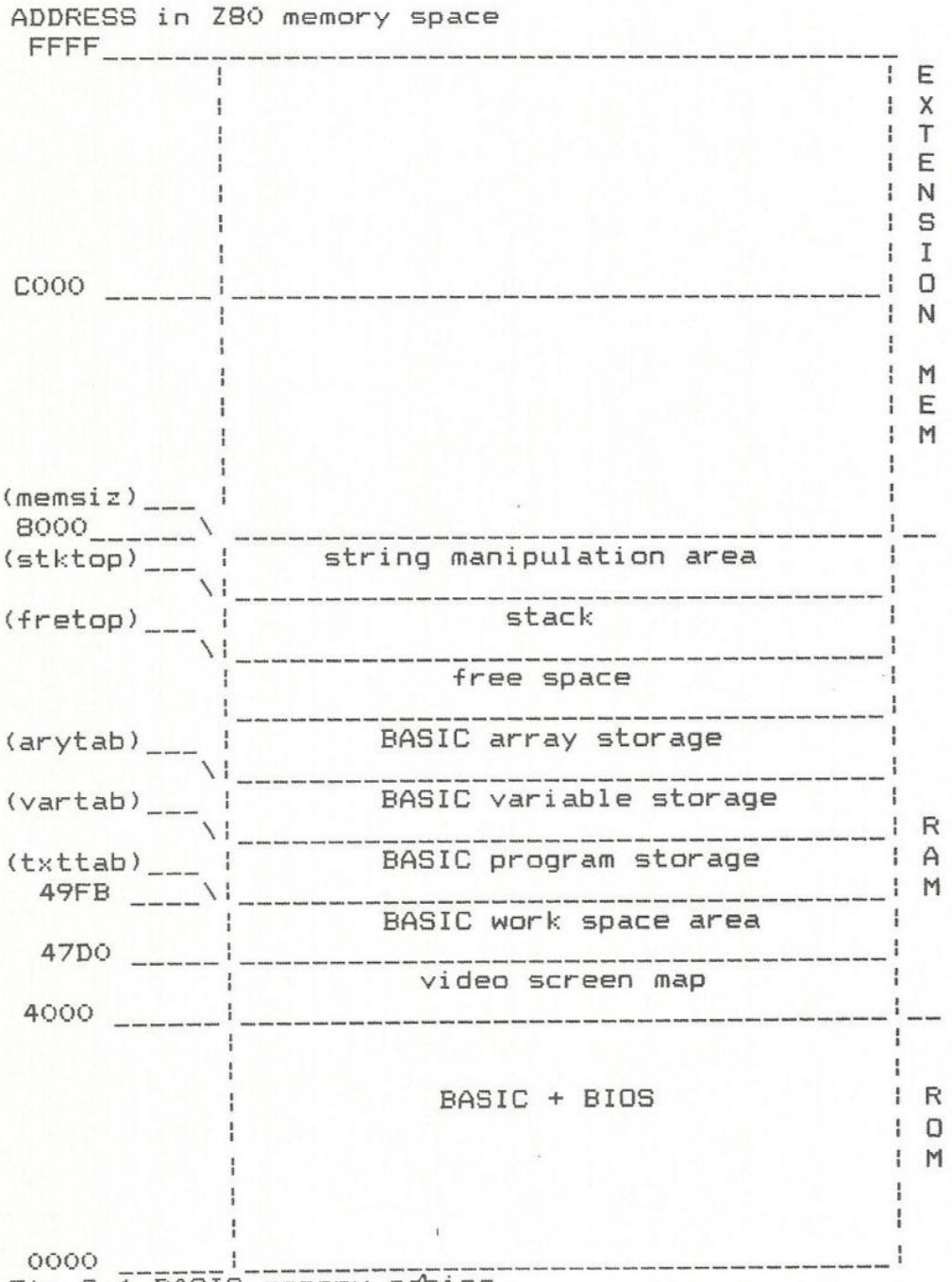


Fig 2.1 BASIC memory mapping

2.1 General Description

The VG5000 hardware does not support a full bank switching mechanism, hence the maximum memory allocation space is 64K. There is, however one exception to this. Because the onboard ROM is capable of being disabled by a signal from the external bus it is posible to map in 48k externally, hence a 64K RAM machine is possible.

2.2 ROM Area

The BASIC and BIOS programs reside in the 16K ROM shown on figure 2.1. There are 14 source modules within this program, which are allocated at the following addresses:

ADDRESS	MODULE	DESCRIPTION
0000	in	interrupt handler and low level comms.
02FF	f4i	BASIC floating point mathematics
0A75	gr	BASIC graphics handler
1000	biinit	BASIC initialization
11F9	txtfil	text messages
1945	bicset	BASIC high level cassette handler
2000	bintrp	BASIC interpreter
2AD4	te	text editor
2EAD	bimisc	BASIC interpreter miscellaneous
3278	co	screen communications
3609	bistrs	BASIC string handler
38D0	biptrg	BASIC pointer get routines
3A59	caset	low level cassette handler
3BD0	bio	BASIC input/output controller
3CD2	biprtr	printer controller

Descriptions of the exact functioning of these modules is described in the source listings and in some cases, later on in this manual.

2.3 RAM Area

RAM area is dynamically allocated by BASIC initialization on power up. The system will search for the existance of RAM from &"4000" up to &"FFFF" and will, assuming all the memory is working, stop at &"7FFF" in the basic console.

The area between &"4000" and &"49FB" is used by the system for screen buffer and work space. The video screen map is used by BASIC for all communications to the video system. All characters written and read back from the screen by BASIC go to this buffer and the buffer is transfered to the video memory as a complete block. The buffer is logically separated into 25 rows of 80 bytes. Each character displayed on the screen consists of 2 bytes, the first is the character code and the second the attribute. For full description of this sequence see Thompson-Efcis EF9345 users manual, Appendix xx. A description the memory locations in the BASIC work area is given below.

The stack pointer is placed below the string space as shown in the memory map figure 2.1 by BASIC. It is possible to move the stack by allocating more string space and presetting the top of memory by the use of the BASIC "CLEAR" command.

BASIC will put program lines into the memory buffer above the BASIC work space area and will put variable storage above the program. A complete description is shown in the RAM allocation map shown below.

The BASIC command "FRE(0)" indicates the number of bytes between the top of variable space and the current stack pointer -96 levels.

The following table gives the RAM locations used by BASIC and BIOS. The description given indicates the primary purpose of the location, however in many cases other routines use locations which are mutually exclusive with the primary function. Identifying secondary uses of memory locations must be done by examination of the source listing since they are too numerous to mention.

In the case of applications programs which do not require the full functions of BASIC and BIOS it is possible to use these allocated memory areas observing the case mentioned above.

2.4 BASIC and BIOS RAM allocation table

ADRS hex	LEN dec	NAME	DESCRIPTION
4000	7D0	screen	video screen map of 25 lines * 80 bytes.
			HOOK VECTORS are programmed to RET on power up. Can be reprogrammed by user with a JP to modify action.
47D0	3	inthk	start of maskable interupt interupt routine
47D3	3	calhk	call routine vector
47D6	3	sonhk	start of sound generator routine
47D9	3	plyhk	start of play routine
47DC	3	rsthk	programmable restart &"30" vector
47DF	3	prthk	start of PRINT command routine
47E2	3	outhk	start of outdo (character output) routine
47E5	3	crdhk	start of crdo (carriage ret/line feed) routine
47E8	3	inlhk	start of inlin (character input) routine
47EB	3	inphk	start of INPUT command routine
			HOOK VECTORS for external devices. Initialize to to JP [adrs]
47EE	3	nmihk	start of non-maskable interupt routine
47F1	3	lpnhk	light pen
47F4	3	dskhk	disk interface
47F7	3	modhk	modem interface
			SCREEN COMMUNICATIONS FLAGS note IX register always points here
47FA	1	intdiv	interupt divider counter for screen update
47FB	1	intact	flag to indicate if screen update required
47FC	1	intrat	interupt divider reload value (ie rate)
47FD	1	cursor	cursor params and border color (MAT reg 9345)
47FE	1	fklock	keyboard lock key status
47FF	1	crchar	most recently scanned keyboard value
4800	1	reptim	key auto repeat timer (incremented every 20mS)
4801	1	repena	key auto repeat flags
4802	1	attcar	forground attribute for character to be printed:- bits 0-2 forground color bit 3 flashing bit 4 double height (alpha) bit 5 double width (alpha) bit 6 reverse video (alpha) bits 4-6 background color (graphic) bit 7 0=alpha 1=graphic
4803	1	attbak	background attribute used by INIT command bits 0-2 forground color of col 0 bit 3 =0 bits 4-6 background color for following alpha chars. bit 7 =1
4804	1	extenf	flag to indicate extension character print mode bits 0-6 =0 bit 7 =0 ROM chars, =1 RAM chars

```
4805 1    xcurso    horizontal position of cursor on screen
4806 1    ycurso    vertical position of cursor on screen
  TEXT EDITOR variables and flags
4807 2    prelin    previous line number to one just located
4809 2    homeln    line number from which home key will list
480B 2    retadr    address of line currently being sent to BASIC
480D 1    entstt    flag to indicate line being sent to BASIC
  SOUND SYSTEM variables
480E 2    sonsav    saves high and low time of current note
  CASSETTE buffers and variables
4810 30   ft        file table for cassette files
482E 1    lowlim    cassette calibration
482F 1    winwid    cassette calibration

                        Module BIINIT

    BASIC work space all bytes initialized on power up
4830 3    ramlow    jump vector to warm start address
4833 3    usrloc    USR(0) function jump vector
4836 15   fdivc     floating point division subroutine work area
4845 41   rndcnt    BASIC random number generater tables
486E 1    lptpos    current line printer column number
486F 1    prtflg    character output stream flag
                        &"00"=terminal output
                        &"01"=printer output
                        &"FF"=cassette output
4870 1    getflg    character input stream flag
                        &"00"=keyboard input
                        &"FF"=cassette input
4871 1    picflg    picture display control flags
                        bit 7  1=scroll disabled
                        bit 6  set by cursor on page end
                        bit 5  1=INPUT statement active
                        bit 4
                        bit 3
                        bit 2  1=continuation permitted
                        bit 1  1=line being entered to BASIC
                        bit 0  continuation char reqd  on  line
                               wrap round
4872 1    cascom    cassette communication flag
                        bit 7  aborted (set by system)
                        bit 6  sumcheck error (set by system)
                        bit 5  verify error (set by system)
                        bit 4  file too big (set by system)
                        bit 3  return if error (set by user)
                        bit 2  0=1200 baud (set by user)
                               1=2400 baud (set by user)
                        bit 1  suppress L/H/H ft (set by user)
                        bit 0  suppress messages (set by user)
  PRINTER flags and variables
4873 1    rawprt    printer description flag
                        &"00" MSX printer (set by user)
                        &"01" foreign printer (set by user)
                        &"ff" no controls (set by user)
4874 1    prtstt    printer status flag
```

4826 2 low — 0 bit half-cycle time
4828 2 high — 1 bit half-cycle time
482A 1 leader — cycles to write/256

with VG5000 character set

including MSX printers with MSX character set

```
                                bit 7  user off (set by PRT key toggle)
                                bit 6  initialized (print string sent)
                                bit 0  busy
4875 1     prtcom     printer communications flag (set by user)
4876 2     prtint     address of printer initiailization string vector
4878 2     prtxlt     address of printer translation table
  TEXT EDITOR miscellaneous
487A 10    contbl     numeric control key codes
4884 1     autflg     flag to indicate AUTO command active
4885 2     autlin     current auto line number
4887 2     autinc     current auto increment
4889 1     allflg     must be 0 to execute BASIC control loop
488A 1     linlen     line length
488B 1     clmlst     position of last comma column
  BASIC miscellaneous
488C 2     curlin     current executing BASIC line number
488E 2     txttab     pointer to beginning of BASIC program store
4890 1     frgflg     0=French messages 1=international
4891 2     kbdtbl     address of keyboard lookup table
4893 1     tmpsav     tempo save in PLAY statement
4894 1     octsav     octave save in PLAY statement
4895 2     stktop     address of top location used by stack
```

Module BINTRP

```
4897 1     bufmin     used by INPUT statement
4898 129   buf        buffer used for crunching inputted lines
                      and executing immediate statements
4919 1     endbuf     end of crunch buffer
491A 1     dimflg     indicates when DIM is active
491B 1     valtyp     the type indicator 0=numeric 1=string
491C 1     dores      flag to disable crunching
491D 2     contxt     save the text pointer after constant
491F 2     memsiz     highest location in memory
4921 2     temppt     pointer at first 3 temp descriptor
4923 120   tempst     storage for numtmp temp descriptor
499B 40    dsctmp     string functions build answers here
49C3 2     fretop     address of top of string free space
49C5 2     temp3      address of end of string erase
49C7 2     temp8      used by garbage collection
49C9 2     endfor     saved text pointer of FOR statement
49CB 2     datlin     data line number
49CD 1     subflg     used in FOR statement
49CE 1     flginp     flags if we are doing input or read
49CF 2     temp       temporary for statement code
49D1 1     ptrfrg     used by RENUM statement
49D2 2     temp2      formula evaluator temp
49D4 2     oldlin     old line number
49D6 2     oldtxt     old text pointer
49D8 2     vartab     pointer to start of simple variable space
49DA 2     arytab     pointer to start of array table.  Incremented
                      by 6 with each simple variable.
49DC 2     strend     end of storage in use
49DE 2     datptr     pointer to data in DATA statement
49E0 2     prmnam     name of parameter that is active
```

49E2	4	prmval	value of active parameter
49E6	3	faclo	low bytes of floating point accumulator
49E9	2	fac	floating point accumalator
49EB	13	fbuffr	buffer for fout
49F8	1	fmltt1	temp for fmult
49F9	2	fmltt2	temp for fmult

2.5 RAM Allocation for Applications Programs

The RAM space available for applications programs is clearly dependent on the method of loading the software into the memory space and the degree of interaction with BASIC. The following is intended only as a guideline to the applications programmer with the following types of system organization and may require additional research using the source listings for bug-free results.

BASIC only from cassette. Leave RAM allocation as specified above and only use space that BASIC allocates. If the program is to be protected from user examination and input is inhibited then the crunch buffer (buf) is available as a 128 byte workspace.

BASIC with machine code subroutines from cassette. Allocation is similar to the situation with a BASIC only application but space must be allocated for the machine code section via use of the BASIC CLEAR statement. For details on how to save and load a program in this format refer to cassette section.

BASIC only and BASIC with machine code subroutines from extension ROM. Point the BASIC program text pointer (txttab) to the start of BASIC program text in ROM and point the simple variable pointer (vartab) and the array table pointer (arytab) to available space in RAM. Leave BASIC work space intact and otherwise use any available memory. Note that (txttab-1) must be made zero.

Machine code only from cassette using BASIC and BIOS features. All memory space from (txttab) to stack bottom is available for program storage and certain locations inside the BASIC work space, such as the crunch buffer (buf) can also be used. It is obviously necessary to check with the BASIC program listing to find use of memory by the routines you require. It is also possible to use the screen buffer providing that the communications procedure to the screen is re-configured.

Machine code only from cassette without BASIC and cassette BIOS features. In this case it is probably safe to use all BASIC work space except the parts concerned with screen communication if the existing screen communications system is used.

Machine code only from ROM. Similar to the above situations but

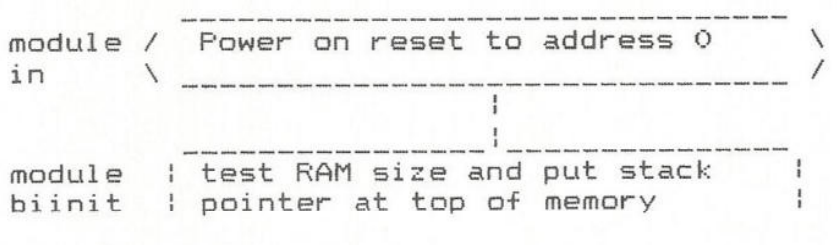
less care need be taken with the cassette locations.

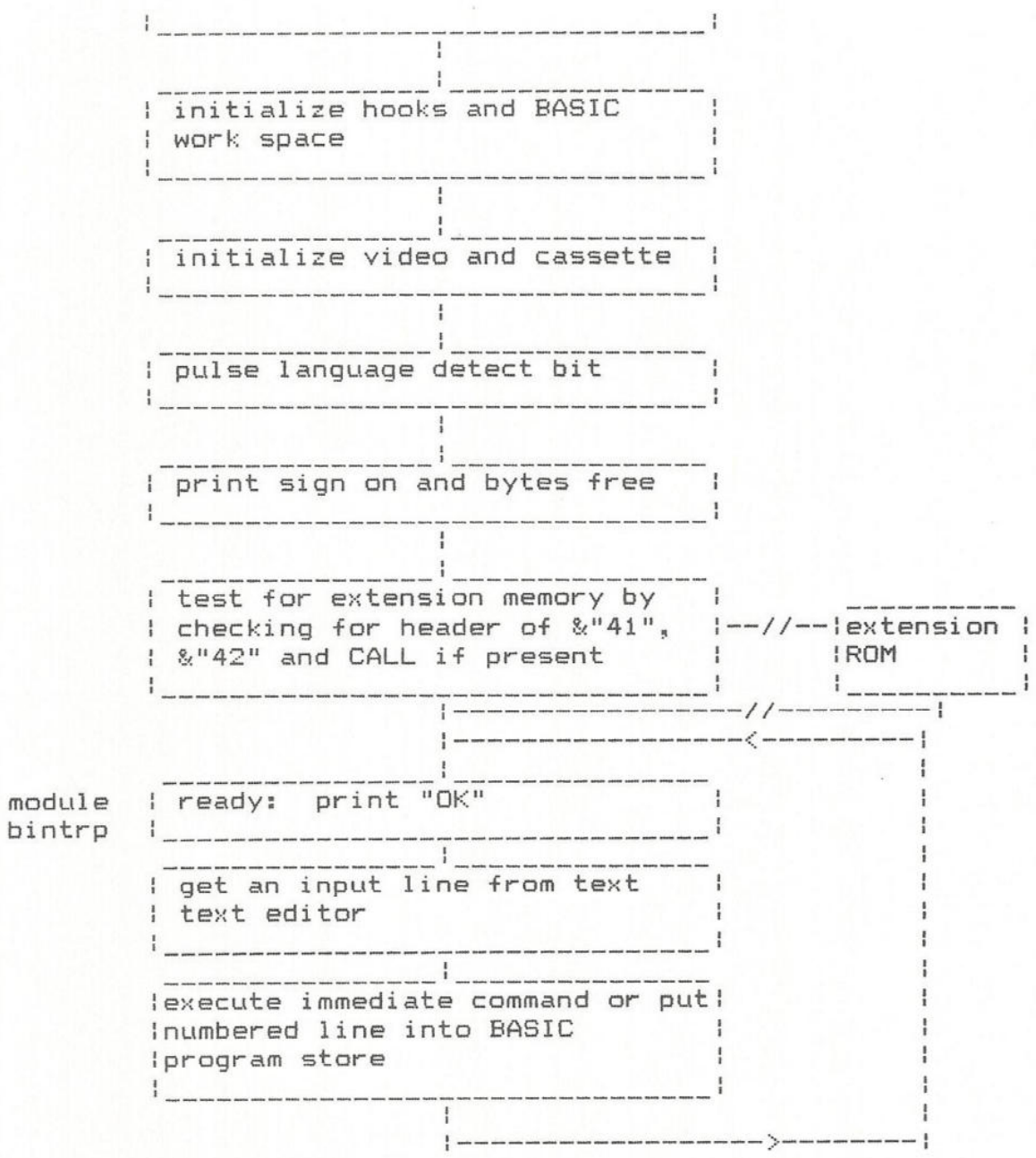
2.6 Uses of Hooks.

Hooks are 3 bytes RAM locations which can be used by the applications programmer to modify the action of existing BASIC and BIOS features. They are initially programmed to RET's since the BASIC and BIOS routines using them perform CALL's to the hook location. To find out exactly how to use the hook, refer to the BASIC program listing.

3 OPERATION OF BASIC

figure 3 Overview of BASIC




initialize hooks and BASIC work space
initialize video and cassette
pulse language detect bit
print sign on and bytes free
test for extension memory by checking for header of &"41", &"42" and CALL if present
extension ROM
module bintrp
ready: print "OK"
get an input line from text text editor
execute immediate command or put numbered line into BASIC program store

Operation of BASIC

3.1 Introduction

Figure 3 shows the general structure of the BASIC interpreter software. The initialization sections are shown in some detail but the BASIC interpreter control loop is shown very greatly simplified. It is not within the scope of this report to go into an indepth analysis of the BASIC interpreter. We will, however investigate the initialization procedure as far as it is relevent to applications software and look in detail at the communications system and the cassette handling system.

3.2 Initialization

The format for BASIC initialization is shown accurately in figure 3 and the source listings show the exact procedure.

Memory checking is performed by uploading the contents of ROM into the memory space from &"4000" to &"FFFF" and comparing the contents.

Some of the hook vectors are initialized to the RET code since they hooks are intended to be used as vectors to subroutines. The hook vectors which are intended for add-on devices are initialized to jump to an error routine since they are not supported in the BASIC console.

Video and cassette systems are then initialized prior to the enabling of interupts since it is the video chip which provides the interupt to the Z80.

The initialization table in module biinit is uploaded to the memory at this point.

Both versions of the VG5000 (ie French and International) use the same ROM. Detection by software of which version is currently active is done by sending a pulse to an I/O port address &"EF" which in the case of the Internal version is connected via a diode on the circuit board to the non-maskable interupt on the Z80. The interupt handler, address &"66", will program a flag in memory which has previously been initialized as 0 to 1 and the keyboard lookup table address which has previously been initialized at the French table to the International table.

This is followed by initialization of some of the BASIC parameters and then a test for external ROM software is made by checking at addresses &"8000", &"A000", &"C000" and &"E000" for

the presence of the header bytes &"41", &"42". If the header is found a CALL will be made to the address held in the 2 bytes following the header. The user can then choose to reprogram hooks to bring in additional BASIC facilities etc and then return to the main control loop with a RET statement or can simply discard the return address on the stack and continue with external software without the intervention of BASIC.

3.3 Main control loop

There are in fact 2 main control loops in BASIC.

The first is based around the routine in bintrp starting at label "main" and is responsible for requesting keyboard entry and line assembly and then determining whether inputted lines are immediate commands (without line numbers) or program lines (with line numbers). The inputted line is compacted so that BASIC keywords are crunched into a single byte token and program lines are then linked into the program storage area so that the lines always come in a line number sequenced order. Immediate command lines however remain in the crunch buffer and the BASIC interpreter then starts analysis of this information and processes the information via the second main control loop.

The second main control loop is based around the routine in bintrp starting at the lable "newstt" which means new statement fetcher. Its function is to process in sequence the program data and when it encounters a tokenized keyword (which it recognizes by having bit 7 set) to locate the address the routine representing that particular tokenized statement or function and execute it. The addresses of these statements and functions can be found near the beginning of the bintrp module in a table called the dispatch table. The routines to which control is dispatched can be in one of the many modules within BASIC and are usually grouped into logical areas (ie: graphics commands are vectored to the graphics handler module "gr").

4 COMMUNICATIONS

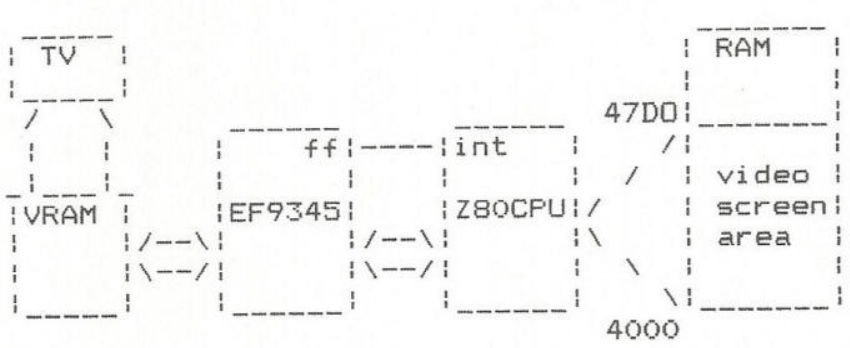


Figure 4 Screen Communications

4.1 General Description

The chain of command for screen communications filters down through several modules. At the top in the module bintrp a call is made to a routine in BIO to request a line input. The inline routine makes successive calls to the text editor module which supplies mostly null codes back which are discarded. When the text editor is active it requests keyboard data from the keyboard handler in module "in" and if it detects a key will either echo a character back to the screen at the cursor position or process a control. Control keys such as screen re-display key and cursor up and down keys are processed in the text editor since they require re-display of the program text. Other control keys are vectored via a lookup table in the module "co". After any change of the screen contents or a cursor movement the flag "intact" is set to indicate that a screen update has been made. When the user presses the "enter" key the text editor will return the data on the entire line back to the inline routine which then compacts the buffer of data to return to bintrp.

4.2 Screen updating

With the exception of DRCS character shape programming there is only one type of transfer to the video system which is performed by the interupt handler routine.

The transfer is synchronized to the start of frame blank by the connection in hardware between frame blank coming from the video chip and interupt on the Z80. This period is chosen because during frame blanking there is the longest available transfer window for communications with the video chip. This transfer which is 2000 bytes long takes approxemately 55mS to complete which is about 2.5 frame times of 20mS per frame. To have a transfer rate so that the screen is sent every available interupt would cause a serious reduction in the speed at which the Z80 proccesses non-interupt software. Therefore an interupt rate sub-divider "intdiv" is used which is reloaded at countdown with the interupt rate preset. This is found in practise to be visually acceptable for mainly text output with a default rate of 20. For fast animation programs it is advisable to either synchronize the screen update using the SCREEN command or by increasing the interupt acceptance rate by using the DISPLAY command whose parameter is loaded into the interupt rate preset.

4.3 Individual Characters

In addition to the inbuilt BASIC facility for whole screen updates it is also possible to send individual characters via a routine that is resident in ROM but is never used by BASIC. For exact details on how to use this facility see chapter 6 BIOS functions.

The ability to send individual characters gives the applications programmer the option of being able to either use the 2000 byte screen buffer area for his own workspace, ☛ If the screen buffer is used in addition to individual character transfers a useful animation feature can be obtained.

It is possible to describe the back-plane of an animated picture inside the screen buffer and have the moving objects displayed as individual characters following the redisplay of the screen. This removes one of the major problems with animation without sprites because it leaves the programmer free from the problem of what to do with the space behind an object that has just moved.

4.4 Other communications methods

The Thomson-Efcis EF9345 video controller chip has a number of modes of operation of which the method used by BASIC ie 16 bits per character is only 1. Other methods of communication and use of the video chip are possible by the applications programmer but it is unlikely that the internal communication routines will be of any use in this situation. Refer to the EF9345 manual for details of how to use other modes of operation.

5 CASSETTE SYSTEM

BYTE No.	NAME	VALUE
00	File type	
01 to 06	File name	(ASCII)
07	Version number	00
08 to 12	Start line number	(ASCII)
13	Protection byte	00=prot
14 to 15	Position to calc sumcheck on input	00,00
16 to 17	Start address	(hex)
18 to 19	Data length	
20 to 21	Sumcheck	

Fig 5 Cassette file table structure

5.1 Introduction

The cassette system is a major part of the overall machine and is probably of great importance to the applications programmer. The VG5000 manual ~~and its predecessor C7420 manual~~ explains in detail how to use the cassette loading system. This section is intended only as a guide to some of the more advanced features of the cassette loading system.

5.2 Software Protection

(13)

Figure 5 shows the structure of the cassette file system. Included in this file header is a byte which is intended to be used for cassette data protection. If it is set to a non-zero value prior to the recording of the tape then when that tape is reloaded it will have the following effect on BASIC execution. A BASIC program and any machine code program will run unless it is halted or there is a forced break such as an error condition etc. At this point BASIC will attempt to return to the main control loop and if it finds the protection bit to be non-zero will perform a restart from zero which will destroy the previous contents of memory. It is also inhibited to do any listing or cassette resaving operation. Therefore it is difficult (although not impossible) for a user to examine or resave a protected program. On each CLOAD, the new value of the protection byte is ORed with the previously held value. Thus a program can ensure that it has been CLOADed by the correct calling program. For example, program 1 can set the byte to &"01". If program 2 sets it to &"02", the value found in byte 13 should be &"03" at program 2 execution time.

"Non-standard" baud rates can be used if required.