

[text of front cover]

MECHATRONIC COMPUTER PRODUCTS

***REFERENCE MANUAL
FOR MECHATRONIC
EXTENDED BASIC II Plus***

**SUPPLEMENTARY VOLUME TO THE TI EXTENDED BASIC REFERENCE MANUAL
FOR THE TEXAS INSTRUMENTS TI-99/4A HOME COMPUTER**

Table of Contents

1. Introduction	1
1.1. Standard Extended BASIC	1
1.2. Extended Statement Set	1
1.3. High Resolution Graphic statements	1
1.4. Standard Mode and Graphic Mode	2
2. Extended Statement Set	5
2.1. BHCOPY	6
2.1.1. Format	6
2.1.2. Description	6
2.1.3. Examples	6
2.2. VPEEK	7
2.2.1. Format	7
2.2.2. Description	7
2.2.3. Examples	7
2.3. VPOKE	8
2.3.1. Format	8
2.3.2. Description	8
2.3.3. Examples	8
2.4. GPEEK	9
2.4.1. Format	9
2.4.2. Description	9
2.5. ALLSET	10
2.5.1. Format	10
2.5.2. Description	10
2.6. WAIT	11
2.6.1. Format	11
2.6.2. Description	11
2.7. MOVE	12
2.7.1. Format	12
2.7.2. Description	12
2.7.3. Examples	13
2.8. MSAVE	14
2.8.1. Format	14
2.8.2. Description	14
2.8.3. Examples	14
2.9. MLOAD	15
2.9.1. Format	15
2.9.2. Description	15
2.10. BYE	16
2.10.1. Format	16
2.10.2. Description	16
2.11. RESTORE	17

2.11.1. Format	17
2.11.2. Description	17
2.12. QUITOF — QUITON	18
2.12.1. Format	18
2.12.2. Description	18
2.13. SPROF — SPRON	19
2.13.1. Format	19
2.13.2. Description	19
2.13.3. Examples	19
2.14. SCREENOF — SCREENON	20
2.14.1. Format	20
2.14.2. Description	20
2.15. FIND	21
2.15.1. Format	21
2.15.2. Description	21
2.15.3. Examples	21
3. High-resolution Graphic Mode	22
3.1. Description	22
3.2. Working in Graphic Mode	23
3.3. Loading the graphic instruction set	25
3.4. Error messages	25
3.5. Graphic statements	26
3.6. APESOFT	27
3.6.1. Format	27
3.6.2. Description	27
3.7. CLRAPE	28
3.7.1. Format	28
3.7.2. Description	28
3.8. GRAFIC	29
3.8.1. Format	29
3.8.2. Description	29
3.8.3. Examples	30
3.9. BYEBYE	31
3.9.1. Format	31
3.9.2. Description	31
3.10. WINDOW	32
3.10.1. Format	32
3.10.2. Description	32
3.10.3. Examples	33
3.11. SETBLE	34
3.11.1. Format	34
3.11.2. Description	34
3.12. CLTBLE	35
3.12.1. Format	35

TEXAS INSTRUMENTS
HOME COMPUTER

3.12.2. Description	35
3.13. TABLE	36
3.13.1. Format	36
3.13.2. Description	36
3.14. SETCOL	37
3.14.1. Format	37
3.14.2. Description	37
3.14.3. Examples	39
3.15. INVERT	40
3.15.1. Format	40
3.15.2. Description	40
3.16. CLSCRN	41
3.16.1. Format	41
3.16.2. Description	41
3.16.3. Examples	41
3.17. CENTRE	42
3.17.1. Format	42
3.17.2. Description	42
3.17.3. Examples	43
3.18. SETTO	44
3.18.1. Format	44
3.18.2. Description	44
3.19. RESET	45
3.19.1. Format	45
3.19.2. Description	45
3.20. IFSET	46
3.20.1. Format	46
3.20.2. Description	46
3.20.3. Examples	47
3.21. MOVE	48
3.21.1. Format	48
3.21.2. Description	48
3.22. REMOVE	49
3.22.1. Format	49
3.22.2. Description	49
3.22.3. Examples	49
3.23. MOVETO	50
3.23.1. Format	50
3.23.2. Description	50
3.24. REMVTO	51
3.24.1. Format	51
3.24.2. Description	51
3.24.3. Examples	51
3.25. TURN	52
3.25.1. Format	52

3.25.2. Description	52
3.26. TURNTO	53
3.26.1. Format	53
3.26.2. Description	53
3.26.3. Examples	53
3.27. RECT	54
3.27.1. Format	54
3.27.2. Description	54
3.28. CLRECT	55
3.28.1. Format	55
3.28.2. Description	55
3.28.3. Examples	55
3.29. CIRCLE	56
3.29.1. Format	56
3.29.2. Description	56
3.30. CLCRCL	57
3.30.1. Format	57
3.30.2. Description	57
3.30.3. Examples	57
3.31. ARCUS	58
3.31.1. Format	58
3.31.2. Description	58
3.32. CLARCS	59
3.32.1. Format	59
3.32.2. Description	59
3.33. ELLIPS	60
3.33.1. Format	60
3.33.2. Description	60
3.34. CLLIPS	61
3.34.1. Format	61
3.34.2. Description	61
3.34.3. Examples	61
3.35. VALUES	62
3.35.1. Format	62
3.35.2. Description	62
3.35.3. Examples	62
3.36. AXIS	63
3.36.1. Format	63
3.36.2. Description	63
3.36.3. Examples	63
3.37. HSTDIA	64
3.37.1. Format	64
3.37.2. Description	64
3.37.3. Examples	64
3.38. CRCDIA	65

TEXAS INSTRUMENTS
HOME COMPUTER

3.38.1. Format	65
3.38.2. Description	65
3.38.3. Examples	65
3.39. WRITE	66
3.39.1. Format	66
3.39.2. Description	66
3.40. DSPLAY	67
3.40.1. Format	67
3.40.2. Description	67
3.41. ACCEPT	68
3.41.1. Format	68
3.41.2. Description	68
3.41.3. Examples	69
3.42. SHIFT	70
3.42.1. Format	70
3.42.2. Description	70
3.43. GSAVE	71
3.43.1. Format	71
3.43.2. Description	71
3.44. GLOAD	72
3.44.1. Format	72
3.44.2. Description	72
3.44.3. Examples	72
3.45. BHCOPY	73
3.45.1. Format	73
3.45.2. Description	73
3.45.3. CAUTION	73
3.45.4. Format samples	73
3.45.5. Examples	74
3.46. Hardcopy demonstration program	75
4. Appendix	79
4.1. Alphabetic quick reference	79
4.2. Reference list for Mechatronic Extended Basic II Plus	86
4.3. Memory mapping	94
4.3.1. TI-99/4A memory map	94
4.4. ROMs and GROMs	95
4.4.1. GROMs	95
4.5. VDP RAM	95
4.5.1. Notes on VDP RAM use	97
4.5.1.1. Screen Images	97
4.5.1.2. Sprite Attributes	97
4.5.1.3. Character Patterns	97
4.5.1.4. Sprite motion	98
4.5.1.5. Color table	98

4.5.1.6. Crunch Buffer	98
4.5.1.7. Basic Programs	98

1. Introduction

Mechatronic Extended BASIC II Plus is a considerably improved version of TI Extended BASIC and offers enhanced capability and flexibility. About 60 powerful commands and statements have been added.

Mechatronic Extended BASIC II Plus consists of three major elements:

1.1. Standard Extended BASIC

The Standard Extended BASIC is a copy of the original TI Extended BASIC (version 110) and is manufactured under license to Texas Instruments. An extensive description of this powerful programming language can be found in the TI Extended BASIC reference manual.

1.2. Extended Statement Set

Mechatronic Extended BASIC II Plus includes more than 20 new statements. Even a skilled programmer using Standard Extended BASIC cannot achieve the equivalent results of these new statements. In addition, the new statements also allow efficient simplifications in program development and offer a significant increase in processing speed.

Examples include: the hardcopy printer routine, direct access to VDP RAM, moving of memory blocks, and saving and loading of memory blocks to and from external devices, including a cassette recorder.

The Mechatronic Extended BASIC II Plus extended statement set is covered in Part 2 of this manual. An introduction to the use of VDP RAM is in the *Appendix* (Part 4) of this manual.

1.3. High Resolution Graphic statements

High resolution graphic statements allow you to address every pixel on the screen using all 16 foreground and background colors. 40 powerful graphic statements are available to draw lines, circles, rectangles, arcs, circles, etc., to save and load graphics on diskettes, or to make hard copies to a printer. The graphic statements are written in TMS 9900 assembly language and require the TI 32K RAM expansion memory (or equivalent). The high resolution graphic statement set is implemented under license to Apesoft Micro Computer Software, Austria. These statements are discussed in Part 3 of this manual.

Mechatronic Extended BASIC II Plus is fully compatible with TI Extended BASIC or Mechatronic (Standard) Extended BASIC. This means that every program written for TI Extended BASIC (version 110) will work with Mechatronic Extended BASIC II Plus. Owners of Mechatronic (Standard) Extended BASIC can get their modules upgraded by their retailers. This upgrade cannot be applied to original TI Extended BASIC modules.

1.4. Standard Mode and Graphic Mode

Mechatronic Extended BASIC II Plus has two screen output modes, which are called *Standard Mode*, and *Graphic Mode*. Both modes may be used during program execution.

Standard Mode is the default after plugging in the module and choosing MECHATRONIC EXTENDED BASIC from the TI selection screen. All statements, commands, and functions of TI Extended BASIC are available in *Standard Mode*, together with the Mechatronic Extended Statement Set. The 32K RAM expansion is not required, but can be used. In *Standard Mode* the high resolution graphic statements are not available.

Graphic Mode is invoked by:

1. Initializing RAM expansion and transferring the graphic statement set with the command

```
CALL APESOFT
```

before loading or typing in any program.

2. Switching to *Graphic Mode* with the statement

```
CALL LINK( "GRAFIC" , mode )
```

Graphic Mode includes all statements in *Standard Mode* except for those that affect screen output. The reason for this is the different kind of manipulation of screen outputs by the TI-99/4A in *Standard Mode* and in *Graphic Mode*. Statements that affect screen output include all *SPRITE* statements and the *SOUND* statement. A detailed list of all statements in both operating modes can be found in the *Appendix* (Part 4).

Graphic Mode can be quit on two ways

1. Reset to *Standard Mode* with the statement:

```
CALL LINK( "BYEBYE" )
```

This is the only method you should use to return to *Standard Mode*!

2. Program interruption in *Graphic Mode* caused by

```
BREAK, STOP, END, FCTN 4 or a program error
```

TEXAS INSTRUMENTS
HOME COMPUTER

A maximum of 2 opened files is allowed in *Graphic Mode*. Therefore, you should not use CALL FILES.

CALL CLRPE quits *Graphic Mode* and causes a reset to the initial condition, just after selection of MECHATRONIC EXTENDED BASIC from the TI selection screen. However, all stored programs and data will be erased.

2. Extended Statement Set

Mechatronic Extended BASIC II Plus is an extension of TI Extended BASIC. More than 20 new statements provide powerful additional features. Even a skilled programmer using Standard Extended BASIC cannot achieve the equivalent results of these new statements. In addition, the new statements also allow efficient simplifications in program development and offer a significant increase in processing speed.

Features include: Screen Copy to a line printer; as well as Save or Load screen displays or other segments of memory to and from diskette or cassette.

All Extended Statements are available in *Standard Mode*. They can also be used in *Graphic Mode* if they do not affect screen output. A detailed list of all statements in both operating modes can be found in the *Appendix (Part 4)*.

The following system configuration is required for *Standard Mode* :

TI-99/4A + Mechatronic Extended BASIC II Plus

Useful, but not essential:

- + 32K byte RAM expansion
- + Disk system
- + Printer (with graphic capabilities)

CAUTION

Be careful when using a 32K memory expansion unit such as the type manufactured by Boxcar Peripherals. These draw power from the console, which can lead to overheating!

2.1. BHCOPY

2.1.1. Format

```
CALL BHCOPY( "filename" ; "esc-sequence" )
```

2.1.2. Description

This subroutine generates a hardcopy of the screen on line printers operating in bit image mode. Sprites are not output..

filename can be any valid name for the printer device, e.g. "RS232.BA=9600.DA=8.CR" or "PIO.CR". The extension .CR must be used for this subroutine to work correctly. Serial interfaces (RS232) need to use 8 data bits and the appropriate extension ".DA=8". Of course the printer has to be set internally to 8 data bits, too.

You should consult your printer reference manual for the correct *esc-sequence*. BHCOPY sends at the beginning of every line ESC (=CHR\$(27)), followed by the inserted string sequence (up to 10 characters), then the characters CHR\$(0), CHR\$(1) (for 256 following bytes), then 256 bytes, corresponding to the hardcopy of one line, and finally "CR" (=CHR\$(13)). Many line printers will work correctly if *esc-sequence* is "L" or "K".

2.1.3. Examples

```
CALL BHCOPY( "RS232.DA=8.CR" , "L" )  
CALL BHCOPY( "PIO.CR" , "K" )
```

Program examples can be found in the description of BHCOPY and in the section of Hardcopy Demonstration in Part 3.

2.2. VPEEK

2.2.1. Format

```
CALL VPEEK(address, numeric variable list)
```

2.2.2. Description

The VPEEK subroutine allows you to read directly the contents of memory addresses in VDP RAM. The first *numeric variable* of *numeric variable list* is assigned the byte value of VDP memory location *address*. All subsequent *numeric variables* of *numeric variable list* are assigned the byte values at subsequent VDP *addresses*. A single byte can range in value from 0 through 255.

Note: The address range of VDP RAM is 0 through 16383 decimal. Entering higher values can cause the computer to malfunction.

2.2.3. Examples

Character patterns are stored in VDP RAM addresses >03F0 through >07FF (decimal: 1008 through 1919). Each character definition requires 8 bytes. The 8-byte character pattern identifier of the cursor (ASCII code 30) is stored starting at address 1008. It may be read by:

```
100 CALL VPEEK(1008,A,B,C,D,E,F,G,H)
110 PRINT A;B;C;D;E;F;G;H
```

The screen output is 8 decimal numbers:

```
0 124 124 124 124 124 124 124
```

The equivalent hexadecimal values are:

```
00 7C 7C 7C 7C 7C 7C 7C
```

If you use these values in a CALL CHAR statement (with an ASCII code > 31) you will duplicate the shape of the cursor.

You can change the shape of the cursor to a “smiley face”:

```
CALL VPOKE(1008,60,126,219,255,231,189,195,126)
```

Starting at address 1008 in VDP RAM. The eight decimal values starting with 60 will be stored in 8 consecutive locations. These decimal values correspond to the hexadecimal pattern identifiers used in CALL CHAR statements.

Note: The cursor modification is preserved after a program BREAK.

2.3. VPOKE

2.3.1. Format

```
CALL VPOKE(address, numeric value list)
```

2.3.2. Description

The VPOKE subprogram allows you to write byte values directly to addresses in VDP RAM.

The contents of *numeric value list* are written consecutively to VDP memory starting at *address*.

Indiscriminate use of VPOKE can result in a computer malfunction. A system crash may occur, which requires you to switch off your system for a moment, and then on to get it running again. However, this will result in the loss of all program and data, if no back up is available.

2.3.3. Examples

The screen image of 32 x 24 = 768 characters is stored in VDP memory addresses >0000 through >02FF (decimal 0 through 767). The characters are stored with an offset of 96 of their ASCII codes. This means the stored byte is the ASCII code of the appropriate character plus 96.

To poke the word "COMPUTER" to the screen, enter the following command:

```
CALL VPOKE(355,163,175,173,176,1B1,180,165,178)
```

355 is the screen address (0 represents the upper left, 767 the lower right corner of the screen), Beginning with this address the ASCII codes plus 96 of the characters in "COMPUTER" will be consecutively stored. The value 163 is the sum of ASCII code 67 (for C) plus the offset of 96. Similarly for the other values.

More information on VDP RAM use can be found in the *Appendix* (Part 4).

2.4. GPEEK

2.4.1. Format

`GPEEK(address, numeric variable list)`

2.4.2. Description

The GPEEK subroutine allows you to read the contents of GROM addresses in the computer. It is similar to CALL PEEK, which operates on CPU addresses.

More details about GROMs and the Graphics Programming Language (GPL) used in the TI-99/4A can be found in the book *TI-99/4A Intern* written by Heiner Martin. (Verlag für Technik und Handwerk GmbH, Baden-Baden, Germany, 1985.)

2.5. ALLSET

2.5.1. Format

CALL ALLSET

2.5.2. Description

CALL ALLSET resets the characters with ASCII codes 32 through 126 to their initial definitions.

CALL ALLSET is similar to the TI Extended Basic statement CALL CHARSET, but also resets the lower-case characters (ASCII 97-126). This can be interesting to execute, e.g. in programs which use redefined character images.

2.6. WAIT

2.6.1. Format

CALL WAIT(*duration*)

2.6.2. Description

CALL WAIT causes a delay. *duration* can be assigned a value from 0 to 16382. The delay time in seconds is the value of *duration* divided by 60 (50 in most countries outside North America). A value of 1200 represents a delay of 20 seconds on North American consoles.

The delay is terminated if a key is pressed. Values exceeding the allowed range do not produce meaningful time delays or cause an error message.

2.7. MOVE

2.7.1. Format

CALL MOVE (*mode, start address, target address, bytes*)

2.7.2. Description

CALL MOVE allows you to move the contents of memory blocks in RAM. There are four different modes available:

- 1 = Source VDP RAM, destination VDP RAM
- 2 = Source VDP RAM, destination CPU RAM
- 3 = Source CPU RAM, destination VDP RAM
- 4 = Source CPU RAM, destination CPU RAM

A *mode* values less than 1 or more than 4 causes an error message.

CALL MOVE can be used to save the screen. If no assembly language programs are loaded, the screen area can be moved (copied) into Low Memory expansion by:

```
CALL MOVE(2,0,8192,768)
```

The screen can be restored by:

```
CALL MOVE(3,8192,0,768)
```

Indiscriminate use of CALL MOVE may cause the computer to malfunction. The result can be loss of program and data. Parts of a program or variables may also be accidentally overwritten. Detailed knowledge of the CPU and VDP memory maps is essential before using CALL MOVE.

2.7.3. Examples

```
100 REM SAVE DSK1.2073MOVDIS
110 REM ***MOVING DISPLAY***
120 CALL CLEAR
130 CALL VPOKE(1072,233,153,233,137,142,0,0,0,46,48,44,34,220,0,0,0)
140 DISPLAY AT(2,2):"Moved screen segment by"::" CALL MOVE"
145 CNT=0
150 DISPLAY AT(20,2):"That is the power of"::
    " MECHATRONIC"::" EXTENDED BASIC II+"
160 CALL MOVE(1,608,448,160)
170 FOR X=1 TO 450
180 CALL MOVE(1,164,163,445)
190 NEXT X
200 CALL WAIT(200)
210 CNT=CNT+1
220 IF CNT<4 THEN GOTO 150
```

2.8. MSAVE

2.8.1. Format

CALL MSAVE (*"filename", start address, bytes*)

2.8.2. Description

CALL MSAVE saves segments of the CPU RAM contents in program format to an external device.

One use of MSAVE is to store assembly programs on a cassette recorder.

```
CALL MSAVE("CS1", 8192, 8192)
```

saves the complete available RAM segment for assembly language programs. The maximum number of bytes that can be saved is 8192. *filename* can be any valid filename, for example, "DSK1.SCREEN".

2.8.3. Examples

```
100 REM SAVE DSK1.SCREEN
110 CALL CLEAR !Test also without this line
120 REM Define plus mark
130 CALL VPOKE(1072,233,153,233,137,140,0,0,0,46,47,44,34,220,0,0,0)
140 REM Write message
150 CALL HCHAR(10,1,32,160)
160 CALL VPOKE(363,173,164,163,168,161,180,178,175,174,169,163)
170 CALL VPOKE(423,165,184,165,174,164,165,164,128,
162,161,179,169,163,128,169,169,134,135)
180 CALL WAIT(100)
190 REM Save screen image in low mem since MSAVE only saes CPU RAM
200 CALL MOVE(2,0,8192,768)
210 CALL CLEAR
220 PRINT "Screen image will be          saved on diskette..."::
    "...and reloaded now"
230 CALL MSAVE("DSK1.2083SCRN_D",8192,768)
240 REM Get it back
250 CALL MLOAD("DSK1.2083SCRN_D")
260 REM Move screen contents from low mem to VDP RAM
270 CALL MOVE(3,8192,0,768)
280 CALL WAIT(300)
```

2.9. MLOAD

2.9.1. Format

CALL MLOAD("*filename*" ,*mode*)

2.9.2. Description

CALL MLOAD loads program files into CPU RAM that were saved with CALL MSAVE. So CALL MLOAD is just the opposite statement of CALL MSAVE.

filename may be any valid file name.

mode may be omitted in Extended BASIC.

If the numeric value of *mode* > 0, this will auto-start an assembly language program saved in program format. Before auto-start, the VDP RAM will be set up using the Editor/Assembler standard mapping.

CALL MLOAD can load any file in PROGRAM format (also BASIC programs), without generating an error message. However, it only works correctly with assembly files in program format or RAM contents that were saved with CALL MSAVE.

CALL MSAVE and CALL MLOAD require a large area of VDP RAM, which can sometimes cause a memory overflow.

2.10. BYE

2.10.1. Format

CALL BYE

2.10.2. Description

The statement CALL BYE quits the BASIC mode of the TI-99/4A and returns to the TI title screen. CALL BYE has the same function as the command BYE, except that it can be used in programs.

2.11. RESTORE

2.11.1. Format

CALL RESTORE(*numeric variable*)

2.11.2. Description

CALL RESTORE tells the computer which is the next DATA statement to be processed. *numeric variable* contains the line number that the computer will process for the next DATA statement when executing a READ statement.

In standard Extended BASIC the RESTORE statement can only use a numeric value (not a variable), which is not nearly as convenient in programming.

CAUTION

If numeric values instead of numeric variables are used for line numbers in the CALL RESTORE statement, they will not be corrected when using a RESEQUENCE command.

2.12. QUITOF — QUITON

2.12.1. Format

CALL QUITOF
CALL QUITON

2.12.2. Description

CALL QUITOF disables the **QUIT** key (**FCTN =**). This helps you avoid accidentally pressing the key and losing your program and data.

CALL QUITON enables the **QUIT** function. When **QUIT** is active, pressing the **QUIT** key returns you to the TI master title screen.

WARNING

CALL QUITOF is not reset by the operating system. The **QUIT** function will remain off even after executing **BYE** or **NEW**. The **QUIT** function can only be restored by executing **CALL QUITON** or by switching the system off and then on.

2.13. SPROF — SPRON

2.13.1. Format

```
CALL SPROF
CALL SPRON
```

2.13.2. Description

CALL SPROF stops the motion of all sprites at once.

CALL SPRON starts the motion of all sprites at once.

For these functions to work, sprites must have been set in motion using CALL MOTION or CALL SPRITE.

CAUTION

CALL SPROF is not reset by the operating system. It remains in effect even after executing BYE or NEW. The function can only be reactivated by executing CALL SPRON or by switching the system off and then on.

2.13.3. Examples

```
100 REM SAVE DSK1.2133SPRITE
110 REM ***SPRITE STOP - SPRITE GO***
120 CALL CLEAR
130 DISPLAY AT(2,2):"SPRITE STOP AND GO using"::"CALL SPROF and CALL SPRON"
140 A=1
150 CALL CHAR(126,"FFFFFFFFFFFFFFFF")
160 CALL MAGNIFY(2)
170 FOR N=1 TO 4
180 CALL SPRITE(#N,126,2,124,124)
190 NEXT N
200 CALL COLOR(#1,5,#2,7,#3,7,#4,5)
210 CALL LOCATE(#2,124,140,#3,140,124,#4,140,140)
220 CALL WAIT(200)
230 CALL SPROF
240 CALL MOTION(#4,0,124,#3,0,124,#2,0,124,#1,0,124)
250 FOR N=1 TO 5
260 CALL SPRON
270 CALL WAIT(200)
280 CALL SPROF
290 CALL WAIT(200)
300 NEXT N
310 CALL DELSPRITE(ALL)
320 CALL ALLSET
```

2.14. SCREENOF — SCREENON

2.14.1. Format

CALL SCREENOF
CALL SCREENON

2.14.2. Description

CALL SCREENOF switches off the screen display. The contents of VDP RAM are not lost. They are just not displayed on the monitor.

CALL SCREENON reactivates the screen display.

After a program BREAK the screen display will be turned on automatically.

2.15. FIND

2.15.1. Format

```
CALL FIND("get-string","string array"(),return variable)
```

2.15.2. Description

The FIND Subroutine looks in a one-dimensional *string array* for a term assigned to *get-string*. The numeric *return variable* contains the element number, in which the term is found. If the term is not found *return variable* is assigned the value -1.

2.15.3. Examples

The following example demonstrates how fast a 4-digit string will be found in an array with 850 elements, if the complete array has to be tested.

```
100 REM SAVE DSK1.2153FIND
110 REM ***FINDTEST***
120 CALL CLEAR
130 PRINT "PERFORMANCE TEST OF          CALL FIND":
140 PRINT "Loading string array of":"850 elements"
150 DIM A$(850)
160 FOR N=1 TO 850
170 A$(N)="TEST"
180 NEXT N
190 A$(850)="xxxxx"
200 B$="xxxxx"
210 PRINT:"Searching for "&B$&"..."
220 CALL FIND(B$,A$(),B)
230 PRINT:"Found in element";B
240 END
```

3. High-resolution Graphic Mode

3.1. Description

Mechatronic Extended Basic II Plus offers the TI-99/4A owner access to high resolution graphics. Using *Graphic Mode*, the powerful graphic capabilities of the computer can be displayed.

It is possible to address each pixel on the screen using 16 foreground and background colors! 40 powerful graphic routines are available in Mechatronic Extended Basic II Plus.

The graphic copy feature to a dot matrix printer is implemented by software, too. Generation of the graphics is very fast. All of the routines are written in 16-bit TMS9900 assembly code. It is creative and fun to watch *Graphic Mode* at work.

To use *Graphic Mode* the following hardware is required:

TI-99/4A + Mechatronic Extended Basic II Plus module
 + 32 K RAM expansion

Useful but not essential:

+ Floppy disk system
+ Printer (with graphic capabilities) and appropriate interface

WARNING

Be careful when using a 32K memory expansion unit such as the type manufactured by Boxcar Peripherals. These draw power from the console, which can lead to overheating!

3.2. Working in Graphic Mode

The principle underlying *Graphic Mode* statements is very simple. The graphic is a cursor or plotter graphic.

The screen is the drawing sheet. When *Graphic Mode* is enabled, the cursor or drawing pen will appear at position 1, 120 in the drawing window (left bottom corner), ready to start operating along the horizontal axis at an angle of 0 degrees. Simultaneously it is the 0-point of the system of user-defined coordinates.

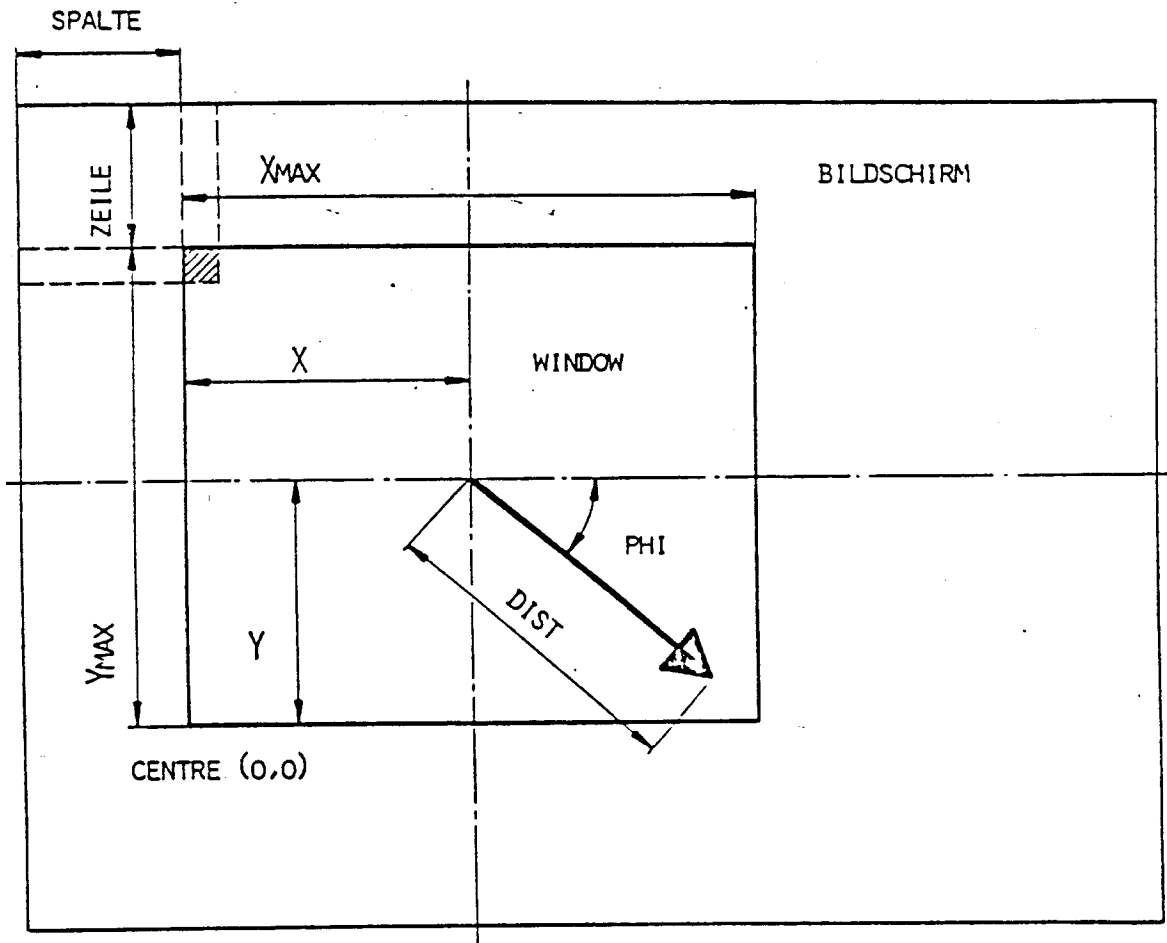
Using simple graphic commands, the cursor is directed across the screen (see fig. 1). All coordinates are identical to mathematical ones.

Lines, circles, squares, ellipses arcs and many other figures can be drawn or erased. Complex geometrical figures, diagrams, and so on can be generated on the screen at very high speed.

All this is available for the TI-99/4A in all colors with a resolution of 256 x 192 pixels. The graphics can also be stored on a floppy disk for later use.

Using *Graphic Mode* you have a powerful programming tool for development of sophisticated programs. The graphic statements are simple and easy to learn.

CONTROLLING THE CURSOR



```
> CALL LINK("WINDOW",ROW,COLUMN)  
> CALL LINK("SETTO",X,Y)  
> CALL LINK("TURNTO",PHI)  
> CALL LINK("MOVE",DIST)
```

Fig. 1. PRINCIPLE OF CURSOR CONTROL

3.3. Loading the graphic instruction set

1. Insert Mechatronic Extended BASIC II Plus Module
2. Switch on disk drives (if available)
3. Switch on 32K RAM Expansion or Peripheral System
4. Switch on Monitor and TI-99/4A
5. Select MECHATRONIC EXTENDED BASIC on selection screen
6. Enter > CALL APESOFT in immediate mode.

The entry of graphic programs is now enabled.

WARNING

Entering

```
> CALL APESOFT
```

after loading any BASIC program will erase the program,

CALL APESOFT closes all opened files, loads the graphic routines into RAM expansion, reserves required space in VDP RAM, and executes a "NEW", deleting the BASIC program in memory.

A maximum of 2 files can be opened *Graphic Mode*. After entering CALL APESOFT, you should not execute a CALL FILES.

3.4. Error messages

Error messages in *Graphic Mode* are not always correct after executing graphic statements,

All conventional errors are trapped correctly by Mechatronic Extended BASIC II Plus. Sometimes the TI-99/4A can fail after an interrupt with **FCTN CLEAR** or by choosing a subroutine. In these cases, you can only recover by switching off the system for a short period.

3.5. Graphic statements

The BASIC control of the high resolution *Graphic Mode* consists of a number of very powerful commands. They simplify the development of programs with complicated graphics tremendously. They are BASIC support commands and have the form:

```
CALL LINK("procedure name",parameter [,parameter,...])
```

CALL LINK connects a BASIC program to an assembly program. *procedure name* calls a certain TMS9900 routine. *parameters* are optional.

Terms in [] can be repeated if desired. Up to 15 parameters can be entered in a single statement.

Depending on the statement, the following parameters can be used:

- Numeric constants
- Numeric variables
- Numeric array elements
- Numeric terms
- String constants
- String variables
- String array elements
- String terms

3.6. APESOFT

3.6.1. Format

CALL APESOFT

3.6.2. Description

The CALL APESOFT command transfers the high resolution graphic statement set to RAM expansion — with reservation it is available and operational.

Please note that CALL APESOFT is a command, which can be only executed in immediate mode.

CALL APESOFT does the following:

- All opened files are closed
- The graphic statements are transferred to RAM expansion.
- Reserves required VDP RAM for graphic executions.
- Executes NEW. This will erase any loaded BASIC program.

After CALL APESOFT is executed, do not use CALL FILES!

3.7. CLRape

3.7.1. Format

CALL CLRape

3.7.2. Description

CALL CLRape restores the initial condition to that in effect immediately after selecting MECHATRONIC EXTENDED BASIC from the TI selection screen. CALL CLRape can only be called after executing CALL APESOFT, otherwise a Syntax Error will result.

CALL CLRape does the following:

- Performs CALL INIT
- Resets VDP RAM
- Closes all opened files
- Executes NEW, erasing the stored program in memory.

3.8. GRAFIC

3.8.1. Format

CALL LINK("GRAFIC" , *mode*)

3.8.2. Description

This command sets the type of graphic mode and initializes all its registers.

A graphic table with maximum 128 vertical and maximum 120 horizontal lines is defined. This table section is now in *Graphic Mode* and every pixel can be individually addressed.

The size restriction of $128 \times 120 = 15,360$ pixels is necessary. If the table were larger, there would not be enough room in VDP RAM for BASIC programs, string variables and so on. To directly address the $192 \times 256 = 49,152$ pixels on a TI screen, together with character and color tables, requires about 12K bytes of VDP RAM. Doing this would overwrite the buffer addresses of the BASIC interpreter.

The graphic table can be placed on the screen at any location. This means you can individually address any of the $192 \times 256 = 49,152$ pixels.

CALL LINK("GRAFIC",*mode*) sets the following internal parameters:

PHI=0	Starting angle of the cursor
TABLEWIDTH	16
FOREGROUND COLOR	Green
BACKGROUND COLOR	Black

Other internal parameters depend on *mode*:

<i>mode</i> = 0	• <i>Graphic Mode</i> (255 rows for graphic) 15 x 16 table rows and columns
X=1, Y=120	• Starting position of the cursor (0,0)-point
<i>mode</i> <> 0	• Text mode (192 rows for graphic) 12 x 16 table rows and columns The commands "DSPLAY" and "ACCEPT" are available for input/output operations.
X=1, Y=88	• Starting position of the cursor (0,0)-point.

CALL LINK("GRAFIC",*mode*) must be the first command before any graphic statement is entered. As soon as this command is executed the standard characters are lost. In addition, the conventional screen I/O commands will not produce the expected results.

TEXAS INSTRUMENTS HOME COMPUTER

CALL LINK("BYEBYE") removes *Graphic Mode* and all I/O statements will work as usual.

When a BREAK (**FCTN CLEAR**) occurs, the program is interrupted and the standard state of the computer is restored.

```
> CONTINUE (or CON)
```

however, does not return to *Graphic Mode*, unless the first command following

```
> CONTINUE (or CON)
```

is

```
CALL LINK "GRAFIC" ,mode)
```

3.8.3. Examples

```
100 REM SAVE DSK1.3083SPIRAL
110 REM ***SPIRAL***
120 CALL LINK("GRAFIC",0)
130 CALL LINK("WINDOW",3,8)
140 CALL LINK("SETTO",64,60)
150 FOR DIST=5 TO 90 STEP 2
160 CALL LINK("MOVE",DIST)
170 CALL LINK("TURN",90)
180 NEXT DIST
190 CALL KEY(0,K,S)
200 IF S=0 THEN 190
```

draws a pale green square-shaped spiral on a black background.

3.9. BYEBYE

3.9.1. Format

```
CALL LINK("BYEBYE")
```

3.9.2. Description

"BYEBYE" removes *Graphic Mode*. It reloads the standard character set and reinitializes *Standard Mode*. The statements SOUND and SPRITE can be used again. The computer works as usual.

Before executing another graphic statement a new CALL LINK("GRAFIC",*mode*) must be issued, otherwise program execution will be interrupted with an error message.

3.10. WINDOW

3.10.1. Format

```
CALL LINK("WINDOW", row, column)
CALL LINK("WINDOW", r, c, x, y, rc, cc)
```

3.10.2. Description

This command transmits sections of the graphic table or the complete graphic table to the screen and has two formats.

The total graphic window (16 x 8 = 128 columns, 15 x 8 = 120 rows) is set on the screen position column (0-32) and row (0-24). The graphic window can be positioned partly or totally outside of the screen (24 rows, 32 columns).

Every graphic window defined before the execution of

```
CALL LINK("WINDOW", row, column)
```

will be deleted if either or both *row* or *column* are negative. Only the absolute values of *row* and *column* are used.

The statement

```
CALL LINK("WINDOW", r, c, x, y, rc, cc)
```

transmits sections of the graphic table to the screen.

The parameters mean:

r,c Screen row (*r*) and column (*c*) at which the upper left corner point of the graphic table is projected.

x,y Upper left corner point of the graphic table where the projection will start.

rc Number of characters of the graphic table in the row direction.

cc Number of characters of the graphic table in the column direction.

If *r* or *c* or are both negative, every graphic window on the screen will be deleted before the new section is drawn.

3.10.3. Examples

```
100 REM SAVE DSK1.3103CIRCLE
110 REM ***CIRCLES***
120 CALL LINK("GRAFIC",0)
130 CALL LINK("WINDOW",1,1)
140 FOR R=2 TO 42 STEP 2
150 CALL LINK("CIRCLE",64,60,R)
160 NEXT R
170 CALL LINK("WINDOW",12,18)
180 CALL LINK("WINDOW",1,19)
190 CALL WAIT(100)
210 CALL LINK("SETCOL",16,5)
220 CALL WAIT(50)
240 CALL LINK("INVERT",1,1,128,120)
250 CALL LINK("WINDOW",4,-8)
260 CALL KEY(0,K,S)
270 IF S=0 THEN 260
```

A number of concentric circles will appear at the left hand top corner of the screen. These will be copied by lines 170 and 180 downwards and to the left and right (tripled). Line 240 will move the circles back to the middle of the screen and the remaining circles are deleted. On its way the graphic changes color and is inverted.

```
100 REM SAVE DSK1.3103PYRMID
110 REM ***PYRAMIDS***
120 CALL LINK("GRAFIC",0)
130 CALL LINK("WINDOW",1,1,1,1,15,16)
140 CALL LINK("WINDOW",1,17,1,1,15,16)
150 CALL LINK("WINDOW",13,1)
160 CALL LINK("WINDOW",13,17)
170 CALL LINK("TURNT0",45)
180 CALL LINK("SETTO",64,48)
190 FOR A=1 TO 36 STEP 2
200 CALL LINK("RECT",A,A,-A,A,-A,-A,A,-A)
210 NEXT A
220 GOTO 220
```

In this example lines 130 and 140 draw top views of four pyramids simultaneously. The upper left window is only a fragment because the line 130 transmits only a section of the graphic table.

3.11. SETBLE

3.11.1. Format

CALL LINK("SETBLE",*width*)

3.11.2. Description

This command dimensions the graphic table.

width Number of columns of the graphic table. It can range from 1 to 32.

191 characters in the text mode and 255 characters in *Graphic Mode* are available for the graphic table.

The height of the graphic table depends on *width* and is calculated by:

height = INT(255 / *width*) for *Graphic Mode*
height = INT(191 / *width*) for text mode

In this way higher and wider graphics can be generated.

CAUTION

A WINDOW statement must follow every SETBLE statement. The WINDOW statement rearranges the screen, otherwise the graphic will be incorrectly generated.

SETBLE also defines the center-point of the system of user-defined coordinates at pixel position:

CENTRX = 1
CENTRY = HEIGHT * 8 = YMAX

3.12. CLTBLE

3.12.1. Format

CALL LINK("CLTBLE")

3.12.2. Description

This command erases the graphic table and the graphic.

But the table sections which are transmitted by WINDOW statements to the screen remain for the input of new graphics.

3.13. TABLE

3.13.1. Format

CALL LINK("TABLE" , *r* , *c* , *xmax* , *ymax* , *bytes*)

3.13.2. Description

This statement returns the present parameters of the graphic table to the following variables:

<i>r</i>	Number of rows of the table
<i>c</i>	Number of columns of the table
<i>xmax</i>	Maximum pixel columns of the table
<i>ymax</i>	Maximum pixel rows of the table
<i>bytes</i>	Number of bytes available for the graphic

Starting with row 1 and column 12 the character bytes in the graphic table are always arranged in ascending order.

The byte number is calculated by:

$$\text{CHAR\#} = (\text{ROW} - 1) * \text{WIDTH} + \text{COLUMN} - 1$$

where:

ROW	= Row of the graphic table
COLUMN	= Column of the graphic table
WIDTH	= Absolute width of the graphic table
CHAR#	= Character byte number of the table

3.14. SETCOL

3.14.1. Format

```
CALL LINK("SETCOL", foreground color, background color)
CALL LINK("SETCOL", n, fg, bg[ ,nl, fgl, bgl, ... ])
```

3.14.2. Description

This command has two formats. It defines foreground and background colors of the graphic.

All the 16 standard BASIC colors can be used either as *foreground color* or *background color*. Several different foreground and background colors can be used simultaneously in one graphic.

If only two parameters are present in the parameter list, *foreground color* and *background color* are altered simultaneously for the entire graphic.

If there are more than two parameters, SETCOL sets character set *n* to foreground color *fg* and background color *bg*.

The character sets for the rows of the graphic windows are defined as following:

8 consecutive bytes construct a character set (0-7, 8-15, ... etc.).

Within the specifications of the above-mentioned table, the foreground and background colors may be defined at random.

Thus a multitude of color combinations is possible. Using a parameter list, up to 5 color sets can be specified.

TEXAS INSTRUMENTS
HOME COMPUTER

Colors for rows and columns of the graphic area (width = 16):

<i>ROWS</i>	<i>COLUMNS</i>	
	<i>1-7</i>	<i>8-16</i>
1	1	2
2	3	4
3	5	6
4	7	8
5	9	10
6	11	12
7	13	14
8	15	16
9	17	18
10	19	20
11	21	22
12	23	24
13	25	26
14	27	28
15	29	30

3.14.3. Examples

```
100 REM SAVE DSK1.3143LEAF
110 REM ***LEAF***
120 RANDOMIZE
130 CALL LINK("GRAFIC",0)
140 CALL LINK("WINDOW",3,8)
150 FOR PHI=0 TO 90 STEP 5
160 CALL LINK("SETTO",64,119)
170 CALL LINK("TURNTO",PHI)
180 CALL LINK("MOVE",1.2*PHI)
190 CALL LINK("TURNTO",180-PHI)
200 CALL LINK("SETTO",64,119)
210 CALL LINK("MOVE",1.2*PHI)
220 NEXT PHI
230 CNT=0
240 CALL WAIT(100)
250 FG=2+14*RND
260 BG=2+14*RND
270 IF BG=FG THEN 260
280 CALL LINK("SETCOL",FG,BG)
290 CALL WAIT(100)
300 CALL LINK("INVERT",1,1,128,120)
310 CNT=CNT+1
320 IF CNT<10 THEN GOTO 240
```

Line 260 defines foreground and background color of the graphic. Line 290 inverts foreground and background color.

3.15. INVERT

3.15.1. Format

CALL LINK("INVERT" , *x* , *y* , *dx* , *dy*)

3.15.2. Description

INVERT swaps the foreground and background colors of a graphic. The following parameters are required:

<i>x,y</i>	Pixel position of the upper left corner of the graphic section which is inverted.
<i>dx</i>	Column pixel position of the section
<i>dy</i>	Row pixel position of the section

Pixels in the graphic which are set are deleted, and vice versa.

3.16. CLSCRN

3.16.1. Format

```
CALL LINK("CLSCRN")
```

3.16.2. Description

This command is similar in effect to CALL CLEAR in BASIC. It erases the graphic. The stored internal cursor parameters remain untouched.

3.16.3. Examples

```
100 REM SAVE DSK1.3163RANLIN
110 REM ***RANDOM STRAIGHT LINES***
120 CALL LINK("GRAFIC",0)
130 CNT=0
140 LCOL=0
150 CALL LINK("WINDOW",-3,8)
160 TMP=INT(RND*16)
170 IF TMP<3 THEN 160
180 IF TMP=LCOL THEN 160
190 LCOL=TMP
200 CALL LINK("SETCOL",LCOL,2)
210 FOR I=1 TO 20
220 X=124*RND
230 Y=120*RND
240 CALL LINK("MOVETO",X,Y)
250 NEXT I
260 CALL WAIT(50)
270 CALL LINK("CLSCRN")
280 CALL WAIT(50)
290 CALL LINK("WINDOW",1,1)
300 CNT=CNT+1
310 IF CNT<10 THEN 150
```

This program example draws 20 successive lines, chooses a random direction and starting from position (1,120) (line 240), clears the graphic (line 270), and starts the graphic again choosing colors at random.

After a short time the statement WINDOW (line 290) shows that only the screen with line 270 has been erased, but the graphic in the table has remained untouched.

WINDOW with a negative parameter (line 150) executes CLSCRN before the section of the graphic table is brought to the screen!

3.17.3. Examples

```
100 REM SAVE DSK1.3173COORDS
110 REM ***COORDINATE SYSTEM***
120 CALL LINK("GRAFIC",1)
130 CALL LINK("WINDOW",7,8)
140 CALL LINK("SETTO",1,1)
150 CALL LINK("RECT",127,-87)
160 CALL LINK("CENTRE",64,44)
170 CALL LINK("AXIS",0,60,60,4,0,40,40,2)
180 CALL LINK("WRITE",8,10,"(0,0)")
190 CALL LINK("DSPLAY",1,5,26,">CALL LINK("CENTRE",64,44)")
200 CALL LINK("DSPLAY",5,5,23,"(-64,+44) (+64,+44)")
210 CALL LINK("DSPLAY",20,5,23,"(-64,-44) (+64,-44)")
220 OPEN #1:"PIO.CR",OUTPUT
230 PRINT #1:CHR$(27)&"A"&CHR$(8)
240 CLOSE #1
250 CALL BHCOPY("PIO.CR","L")
260 STOP
```

This sample program produces the drawing of fig. 2 with an EPSON MX 80 or the similar Texas Instruments Line Printer PHP 2500.

3.18. SETTO

3.18.1. Format

CALL LINK("SETTO", x, y [, $x1, y1, \dots$])

3.18.2. Description

SETTO sets pixels at coordinates row y , column x . On the screen are rows 1 through 120 and columns 1 through 128.

The range of the values has no restrictions. The coordinates may be positive or negative, their values high, low or random, or even floating point numbers. They are internally rounded to the nearest integer number.

Numbers greater than 32,768 and less than -32,767 are displayed incorrectly. There is no error message when this range is exceeded! When a parameter list is used, up to 7 pixels can be defined simultaneously.

The internal angle ϕ of the cursor remains unchanged.

3.19. RESET

3.19.1. Format

CALL LINK("RESET", x, y [, $x1, y1, \dots$])

3.19.2. Description

RESET deletes the pixels with the coordinates x,y . All conditions for SETTO apply to RESET as well.

3.20. IFSET

3.20.1. Format

CALL LINK("IFSET",*x,y,var[,x1,y1,var1,...]*)

3.20.2. Description

This statement checks whether a pixel with the coordinates *x,y* is set and returns the following values in *var*:

Pixel (<i>x,y</i>) set	<i>var</i> =-1
Pixel (<i>x,y</i>) deleted	<i>var</i> =0
Pixel (<i>x,y</i>) outside the graphic window	<i>var</i> =+1

Up to 5 pixels can be checked simultaneously with one parameter list. All conditions for SETTO and RESET apply to IFSET as well.

3.20.3. Examples

```
100 REM SAVE DSK1.3203SINE
110 REM ***SINE***
120 CALL LINK("GRAFIC",0)
130 CALL LINK("WINDOW",3,8)
140 CALL LINK("CENTRE",4,60)
150 PI180=4*ATN(1)/180
160 CALL LINK("AXIS",0,0,118,5,0,50,50,5)
170 FOR PHI=0 TO 360 STEP 2
180 X=PHI/3+20
190 Y=20*SIN(PHI*PI180)
200 CALL LINK("SETTO",X,Y,X,Y*1.5,X,Y*2)
210 NEXT PHI
220 FOR PHI=0 TO 360 STEP 2
230 X=PHI/3+20
240 Y=20*SIN(PHI*PI180)
250 CALL LINK("RESET",X,Y*1.5,X,Y*2)
260 NEXT PHI
270 X=INT(128*RND)+1
280 Y=INT(120*RND)+1
290 CALL LINK("IFSET",X,Y,A)
300 IF A=0 THEN 270
310 CALL LINK("BYEBYE")
320 CALL CLEAR
330 PRINT "POINT X=";X;"Y=";Y;" : IS SET"
340 STOP
```

This program example draws 3 sine curves (170-210), deletes them (220-260), and stays in a holding loop until it has found a set pixel (290-300).

3.21. MOVE

3.21.1. Format

CALL LINK("MOVE",*dist*)

3.21.2. Description

This statement draws a line of length *dist*, starting from the present cursor position with the internal angle *phi*. The position *dist* horizontally and vertically corresponds exactly with the number of pixels; the number of pixels themselves depends on the set angle. After performing *dist* the cursor stops at the end coordinates (last stored pixel). *phi* remains unchanged.

Positive values of *dist* work in the present direction of the cursor, negative values work 180 degrees opposite. *dist* can assume any value, although the range limits are the same as those in the statement SETTO.

3.22. REMOVE

3.22.1. Format

CALL LINK("REMOVE",*dist*)

3.22.2. Description

REMOVE has the same effect as MOVE, but here the pixels from position x,y to the new position *dist* of the cursor are deleted.

3.22.3. Examples

```
100 REM SAVE DSK1.3223STAR
110 REM ***STAR***
120 CALL LINK("GRAFIC",0)
130 CALL LINK("WINDOW",3,8)
140 CALL LINK("WRITE",15,5,"GRAPHIC TEXT")
150 CNT=0
160 FG=3+13*RND
170 CALL LINK("SETCOL",FG,2)
180 CALL LINK("SETTO",112,60)
190 CALL LINK("TURNTO",36)
200 FOR N=1 TO 10
210 CALL LINK("TURN",108)
220 CALL LINK("MOVE",60)
230 NEXT N
240 CALL WAIT(50)
250 CALL LINK("SETTO",112,60)
260 CALL LINK("TURNTO",36)
270 FOR N=1 TO 10
280 CALL LINK("TURN",108)
290 CALL LINK("REMOVE",60)
300 NEXT N
310 CALL WAIT(50)
320 CNT=CNT+1
330 IF CNT<6 THEN GOTO 160
```

Through skilled application of a few commands a star is drawn on the screen, then deleted and then the game starts again in different colors.

3.23. MOVETO

3.23.1. Format

```
CALL LINK("MOVETO",x,y[,x1,y1,...])
```

3.23.2. Description

MOVETO draws a line from the current internal position of the cursor to the position specified by the parameter pair x and y .

The parameter list can hold a maximum of 7 positions. If there are more than two parameters, the line will be drawn from the previous position to the next.

After the execution of MOVETO the cursor remains at the last position in the parameter list. The internal angle and the colors will remain unchanged with MOVETO. Lines can also be drawn outside of the graphic window border.

3.24. REMVTO

3.24.1. Format

```
CALL LINK("REMVTO",x,y,[,x1,y1,...])
```

3.24.2. Description

REMVTO works opposite MOVETO in that lines are deleted. All the conditions for MOVETO also apply to REMVTO.

3.24.3. Examples

```
100 REM SAVE DSK1.3243THREAD
120 REM ***THREAD GRAPHIC***
130 CALL LINK("GRAFIC",0)
140 CALL LINK("WINDOW",3,8)
150 D=2.6
160 Z=125
170 S1=128
180 FOR S=1 TO 126 STEP 5
190 Z=Z-5
200 S1=S1-D
210 CALL LINK("SETTO",S,120)
220 CALL LINK("MOVETO",S1,Z)
230 NEXT S
240 Z=125
250 S1=1
260 FOR S=126 TO 1 STEP -5
270 Z=Z-5
280 S1=S1+D
290 CALL LINK("SETTO",S,120)
300 CALL LINK("MOVETO",S1,Z)
310 NEXT S
320 CALL KEY(0,K,S)
330 IF S=0 THEN 320
```

3.25. TURN

3.25.1. Format

CALL LINK("TURN",*phi*)

3.25.2. Description

This command adds to the present internal angle of the cursor the angle *phi* in decimal degrees.

The limitations of the angle are ± 2047 degrees. Internally the angle is modulated from 0-360 degrees.

The trigonometrical functions are generated by the computer via interpolation tables. Since the storage capacity is very limited the angles are interpolated in steps of 5 degrees. This may lead to inexact results when using intermediate values.

3.26. TURNTO

3.26.1. Format

```
CALL LINK("TURNTO",phi)
```

3.26.2. Description

This command imperatively sets the internal angle of the cursor to *phi* (degrees). All limitations for TURN apply to TURNTO.

3.26.3. Examples

```
100 REM SAVE DSK1.3263OCTAGN
120 REM ***OCTAGONS***
130 CALL LINK("GRAFIC",0)
140 CALL LINK("WINDOW",4,5)
150 CALL LINK("WINDOW",6,19)
160 DIST=2
170 FOR S=28 TO 108 STEP 4
180 CALL LINK("SETTO",S,42)
190 CALL LINK("TURNTO",90)
200 DIST=DIST+2
210 FOR I=1 TO 8
220 CALL LINK("TURN",45)
230 CALL LINK("MOVE",DIST)
240 NEXT I
250 NEXT S
260 CALL KEY(0,K,S)
270 IF S=0 THEN 260
```

3.27. RECT

3.27.1. Format

CALL LINK("RECT",*a*,*b*[,*a1*,*b1*,...])

3.27.2. Description

Starting from its present position and the internal angle of the cursor, RECT draws rectangles with the sequence

$$a \rightarrow b \rightarrow a \rightarrow b$$

The rectangle turns clockwise, if *b* is positive. The example shows the influence of the operational sign of the side length with reference to its ultimate position.

The internal angle and the position of the cursor are not influenced by RECT. The side length of the rectangle can take any value. Up to 7 rectangles can be defined with one parameter list. But they all begin at the same starting position and also finish there.

3.28. CLRECT

3.28.1. Format

```
CALL LINK("CLRECT",a,b[,a1,b1,...])
```

3.28.2. Description

Works identically to RECT, except that CLRECT deletes the rectangles.

All conditions of RECT apply to CLRECT.

3.28.3. Examples

```
100 REM SAVE DSK1.3283RECT
110 REM ***RECTANGLES***
120 CALL LINK("GRAFIC",0)
130 CALL LINK("WINDOW",3,8)
140 CALL LINK("SETTO",64,60)
150 CNT=0
160 A=40
170 B=20
180 CALL LINK("RECT",A,B,A,-B,-A,B,-A,-B)
190 CALL WAIT(50)
200 CALL LINK("CLRECT",A,B,A,-B,-A,B,-A,-B)
210 CALL LINK("TURN",45)
220 CNT=CNT+1
230 IF CNT<9 THEN GOTO 180
```

Line 180 draws 4 rectangles with only one command, Line 200 deletes them. Line 210 turns the internal angle by 45 degrees.

3.29. CIRCLE

3.29.1. Format

CALL LINK("CIRCLE", x,y,r [, $x1,y1,r1,\dots$])

3.29.2. Description

CIRCLE draws a circle with center x,y and radius r . With one parameter list, up to 5 different circles can be drawn.

The parameters can assume any value. For the radius the absolute value is worked out automatically. If the value for $r = 0$, CIRCLE sets a point (pixel). After the execution of CIRCLE the cursor takes the center point of the last drawn circle.

The internal angle ϕ remains unchanged.

Due to internal rounding errors the circular arcs may not appear quite smooth.

3.30. CLCRCL

3.30.1. Format

CALL LINK("CLCRCL",x,y,r[,x1,y1,r1,...])

3.30.2. Description

CLCRCL works identically to CIRCLE, except that the circles are deleted. All conditions for CIRCLE also apply to CLCRCL.

3.30.3. Examples

```
100 REM SAVE DSK1.3303CIRCLE
110 REM ***CIRCLES***
120 PI4=4*ATN(1)
130 CALL LINK("GRAFIC",0)
140 CALL LINK("WINDOW",3,8)
150 CALL LINK("CENTRE",64,60)
160 CALL LINK("CIRCLE",0,0,30)
170 FOR PHI=0 TO 2*PI4 STEP PI4/16
180 CALL LINK("CIRCLE",30*COS(PHI),30*SIN(PHI),30)
190 NEXT PHI
200 CALL KEY(0,K,S)
210 IF S=0 THEN 200
```

3.31. ARCUS

3.31.1. Format

CALL LINK("ARCUS",*x,y,r,phi,dphi*[,*x1,y1,r1,phi1,dphi1*,...])

3.31.2. Description

ARCUS draws circular arcs with the following parameters:

<i>x,y</i>	Center point of the arc
<i>r</i>	Radius of the arc
<i>phi</i>	Starting angle of the arc (absolute)
<i>dphi</i>	Arc angle of the arc

Three arcs can be generated simultaneously with one command. Due to internal rounding errors and generation of the trigonometrical functions via interpolation tables, the results are not always satisfying.

The coordinates of the cursor describe the last drawn arc pixel after the execution of ARCUS.

3.32. CLARCS

3.32.1. Format

CALL LINK("CLARCS", *x, y, r, phi, dphi* [, *x1, y1, r1, phi1, dphi1, ...*])

3.32.2. Description

CLARCS works identically to ARCUS, except that CLARCS deletes all arc pixels.

3.33. ELLIPS

3.33.1. Format

CALL LINK("ELLIPS", x,y,a,b [, $x1,y1,a1,b1,\dots$])

3.33.2. Description

ELLIPS draws ellipses with axis center point x,y major semi-axis a , minor semi-axis b , and inclination ϕ to the major semi-axis.

A maximum of three different ellipses can be drawn with one parameter list. The parameters can assume any values except 0. The absolute value is automatically used for the major and minor semi-axis. After the execution of ELLIPS the cursor assumes the coordinates of the semi-axis points of intersection. The internal angle ϕ remains unchanged.

Through internal rounding errors the elliptical arcs may sometimes not appear quite smooth. This occurs particularly when the main axes are inclined to the horizontal or vertical, because the coordinate transformations are carried out by interpolated trigonometrical functions.

3.34. CLLIPS

3.34.1. Format

```
CALL LINK("CLLIPS",x,y,a,b[,x1,y1,a1,b1,...])
```

3.34.2. Description

CLLIPS works like ELLIPS, except that the ellipses are deleted. All conditions listed for ELLIPS apply to CLLIPS.

3.34.3. Examples

```
100 REM SAVE DSK1.CONE
110 REM ***CONE***
120 CALL LINK("GRAFIC",0)
130 CALL LINK("WINDOW",3,8)
140 CALL LINK("SETCOL",15,2)
150 CNT=0
160 M=0
170 A=42
180 B=22
190 FOR Y=81 TO 1 STEP -8
200 IF M=1 THEN 230
210 M$="ELLIPS"
220 GOTO 240
230 M$="CLLIPS"
240 CALL LINK(M$,54,Y,A,B)
250 A=A-4
260 B=B-2
270 NEXT Y
280 M=M+1
290 CNT=CNT+1
300 IF CNT>5 THEN END
310 CALL WAIT(75)
320 IF M=1 THEN 170
330 M=0
340 GOTO 170
```

If M=1, a cone is always drawn due to the control commands in lines 320-340. M\$, a string variable, can also be passed as *procedure name*.

The program can be terminated by entering **FCTN CLEAR**.

3.35. VALUES

3.35.1. Format

CALL LINK("VALUES",*x*,*y*,*phi*,*fg*,*bg*)

3.35.2. Description

VALUES returns the present internal parameters to the variable list.

<i>x</i>	Cursor column
<i>y</i>	Cursor row
<i>phi</i>	Cursor angle
<i>fg</i>	Foreground color
<i>bg</i>	Background color

As the angle is modulated internally, it is always between 0-360 degrees independent of the previous input.

3.35.3. Examples

```
100 REM SAVE DSK1.3353VALUES
110 REM ***VALUES***
120 CALL LINK("GRAFIC",0)
130 CALL LINK("WINDOW",3,8)
140 FOR I=1 TO 12
150 CALL LINK("SETTO",64,60)
160 CALL LINK("MOVE",30)
170 CALL LINK("VALUES",X,Y,PHI,FG,BG)
180 CALL LINK("CIRCLE",X,Y,20)
190 CALL LINK("TURN",30)
200 NEXT I
210 CALL KEY(0,K,S)
220 IF S=0 THEN 210
```

From the center point of the graphic window, program line 160 draws a line with length 30.

Line 170 determines the final cursor position. Line 180 takes this position as center point for a circle with radius 20.

3.36. AXIS

3.36.1. Format

```
CALL LINK("AXIS",x,lenxr,lenxl,deltax,y,lenyu,lenyd,deltay)
```

3.36.2. Description

AXIS draws a system of coordinates with the following parameters:

<i>x,y</i>	Center point of coordinates
<i>lenxl</i>	Left hand side X-semi-axis (length)
<i>lenxr</i>	Right hand side X-semi-axis (length)
<i>deltax</i>	Pitch of the X-grid
<i>lenyu</i>	Top Y-semi-axis (length)
<i>lenyd</i>	Bottom Y-semi-axis (length)
<i>deltay</i>	Pitch of the Y-grid

All values are taken as absolute values. Any semi-axis with a value of 0 is not drawn.

If the value for the grid equals 0 or more than that of the corresponding semi-axis, no grid will be drawn.

After the execution of AXIS the cursor will assume the position at the center point of the coordinate system.

The internal angle *phi* is altered. The system of coordinates may not be completely on the screen.

3.36.3. Examples

```
100 REM SAVE DSK1.3363ZYKLOM
120 REM ***ZYKLOM***
130 CALL LINK("GRAFIC",0)
140 CALL LINK("WINDOW",3,8)
150 PI2=8*ATN(1)
160 CALL LINK("AXIS",8,0,116,4,60,60,59,4)
170 X=7
180 CALL LINK("SETTO",X,40)
190 FOR PHI=0 TO 3*PI2 STEP PI2/16
200 X=X+2
210 Y=20*(SIN(2*PHI)+2*COS(PHI))+60
220 CALL LINK("MOVETO",X,Y)
230 NEXT PHI
240 CALL KEY(0,K,S)
250 IF S=0 THEN 240
```

3.37. HSTDIA

3.37.1. Format

CALL LINK("HSTDIA",*x,y,width,height,depth*)

3.37.2. Description

HSTDIA draws a block diagram with the following parameters:

<i>x,y</i>	Coordinates of the left bottom corner of the block
<i>width</i>	Width of block diagram
<i>height</i>	Height of block diagram
<i>depth</i>	Depth of block diagram

Only the absolute values are used.

3.37.3. Examples

```
100 REM SAVE DSK1.3373HISTO
110 REM ***HISTOGRAMS***
120 CALL LINK("GRAFIC",0)
130 CALL LINK("WINDOW",3,8)
140 FOR N=2 TO 20
150 CALL LINK("SETCOL",N,14,2)
160 NEXT N
170 CALL LINK("AXIS",8,10,120,8,20,90,0,4)
180 CALL LINK("HSTDIA",16,22,12,80,6)
190 CALL LINK("HSTDIA",40,22,12,45,6)
200 CALL LINK("HSTDIA",70,22,12,67,6)
210 CALL LINK("HSTDIA",94,22,16,12,6)
220 CALL LINK("WRITE",15,3,"HISTOGRAMS")
230 CALL KEY(0,K,S)
240 IF S=0 THEN 230
```

3.38. CRCDIA

3.38.1. Format

CALL LINK("CRCDIA",*x,y,radius,phi,dphi*[,*x1,y1,phi1,dphi1,...*])

3.38.2. Description

CRCDIA draws a circular diagram with the following parameters:

<i>x,y</i>	Coordinates of circular segment center point
<i>radius</i>	Radius of circular segment
<i>phi</i>	Start angle of circular segment
<i>dph</i>	Final angle of circular segment

Only the absolute values are used.

3.38.3. Examples

```
100 REM SAVE "DSK1.3383CIRCDI"
110 REM ***CIRCULAR DIAGRAMS***
120 CALL LINK("GRAFIC",0)
130 CALL LINK("WINDOW",3,8)
140 FOR N=1 TO 13 STEP 2
150 CALL LINK("SETCOL",N,16,2,N+1,14,2,N+14,9,2,N+15,14,2)
160 NEXT N
170 CALL LINK("CRCDIA",64,60,38,180,276)
180 CALL LINK("CRCDIA",68,60,38,276,450)
190 CALL LINK("CRCDIA",68,64,38,90,180)
200 CALL LINK("WRITE",14,2,"CIRC-DIAGRAMS")
210 CALL KEY(0,K,S)
220 IF S=0 THEN 210
```

3.39. WRITE

3.39.1. Format

CALL LINK("WRITE",*r,c,string[,r1,c1,string1,...]*)

3.39.2. Description

WRITE enables the mixing of graphic and text,

At position row (*r*) and column (*c*) of the graphic window WRITE displays *string*.

Limitations: *r* 1 to 15
 c 1 to 16

If *string* is too long for the appropriate line, the rest will be cut. A maximum of 5 strings can be entered with one parameter list. The ASCII codes (upper cases, lower cases, digits and special characters) can be used.

3.40. DSPLAY

3.40.1. Format

CALL LINK("DSPLAY", *r*, *c*, *size*, *var\$*)

3.40.2. Description

This statement corresponds to the standard Extended Basic DISPLAY AT. It uses the following parameters:

<i>r</i>	Row of screen (1-24)
<i>c</i>	Column of screen (1-32)
<i>size</i>	Length of the string (max. 32)
<i>var\$</i>	String variable (max. 32 characters)

With this statement *size* characters of string *var\$* are transmitted to the screen position (*r,c*). Graphic values located under the string are erased (Note the difference to WRITE!).

If *size* is negative, *size* positions are not deleted before the output of the string.

DSPLAY is available in *Graphic Mode* only if the mode is <> 0.

3.41. ACCEPT

3.41.1. Format

CALL LINK("ACCEPT", *r*, *c*, *size*, *var*\$)

3.41.2. Description

This statement corresponds to the standard Extended Basic ACCEPT AT and uses the following parameters:

<i>r</i>	Row of the screen (1-24)
<i>c</i>	Column of the screen (1-32)
<i>size</i>	Length of the string (max. 32 characters)
<i>var</i> \$	String variable (max. 32 characters)

ACCEPT accepts *size* characters of a string at the position (*r*,*c*). If *size* is positive *size* positions are deleted previously.

During the input the following keys are active:

UALPHA	ASCII codes (32-96)
<-	Cursor left
->	Cursor right
ERASE	Deletes the input
ENTER	Accepts the string
REPEAT	Repeat function

CAUTION

FCTN QUIT and FCTN CLEAR are ineffective during the execution of this command.

ACCEPT will accept all key codes, but for text purposes only UALPHAs (upper cases) are useful.

3.41.3. Examples

```
100 REM SAVE DSK1.3413STAR1
110 REM ***STAR1***
120 CALL LINK("GRAFIC",1)
130 CALL LINK("WINDOW",3,8)
140 CALL LINK("SETCOL",12,2)
150 CALL LINK("SETTO",5,60)
160 CALL LINK("MOVE",120)
170 CALL LINK("SETTO",64,1)
180 CALL LINK("TURN",90)
190 CALL LINK("MOVE",120)
200 S1=S
210 Z1=60
220 Z2=36
230 S2=64
240 S3=124
250 Z3=60
260 Z4=84
270 S4=64
280 FOR I=1 TO 10
290 CALL LINK("SETTO",S1,Z1)
300 CALL LINK("MOVETO",S2,Z2,S3,Z3,S4,Z4,S1,Z1)
310 S1=S1+4
320 Z2=Z2-4
330 S3=S3-4
340 Z4=Z4+4
350 NEXT I
360 CALL LINK("WRITE",11,3,"GRAPHIC TEXT")
370 CALL LINK("DSPLAY",17,3,27,"ENTER THE WORD "STOP"")
380 CALL LINK("DSPLAY",18,3,27,"TO EXIT THE PROGRAM")
390 CALL LINK("ACCEPT",22,3,4,M$)
400 IF M$<>"STOP" THEN 120
410 STOP
```

3.42. SHIFT

3.42.1. Format

CALL LINK("SHIFT",*deltax*,*deltay*)

3.42.2. Description

This statement performs a linear transformation of the graphic.

<i>deltax</i>	Movement in X-direction (columns) (positive values rightwards)
<i>deltay</i>	Movement in Y-direction (rows) (positive values downwards)

CAUTION

The graphic windows in the screen will not be transformed!

3.43. GSAVE

3.43.1. Format

CALL GSAVE ("filename")

3.43.2. Description

This statement can be used to save screen displays.

Graphics can only be saved to floppy disk. Any valid *filename* may be used. *filename* can be entered as a string or a string variable. The name of the file may not be longer than 9 characters,

The saved file is formatted in memory image format. The colors of the graphic and the position of the graphic window are stored in separate files for a total of 3 disk files.

3.44. GLOAD

3.44.1. Format

```
CALL GLOAD("filename")
```

3.44.2. Description

This statement loads a graphic called *filename* from diskette into VDP RAM, that was previously stored with GLOAD.

Using CALL GLOAD without executing previous a GRAFIC statement will cause an error message and program execution interrupt.

3.44.3. Examples

```
100 REM SAVE DSK1.3443GSAVE
110 REM ***GSAVE***
120 CALL LINK("GRAFIC",0)
130 CALL LINK("WINDOW",3,8)
140 FOR R=2 TO 40
150 CALL LINK("CIRCLE",64,60,R)
160 NEXT R
170 CALL GSAVE("DSK1.3443GDATA")
180 STOP
```

> NEW

```
100 REM SAVE DSK1.3443GLOAD
110 REM ***GLOAD***
120 CALL LINK("GRAFIC",0)
130 CALL GLOAD("DSK1.3443GDATA")
140 CALL KEY(0,K,S)
150 IF S=0 THEN 140
```

The previous example causes the generation of the following files on DSK1:

3443GDATA Contains graphic table and parameters
3443GDATA1 Contains a screen dump
3443GDATA2 Contains colors of the graphic

By saving and loading the graphic only the first *filename*, in this example 3443GDAT, must be stated. The input of the second or third file name leads to incorrect functions.

The statement WINDOW can be canceled by loading the graphic because every window on the screen is overwritten by the loaded graphic.

3.45. BHCOPY

3.45.1. Format

```
CALL BHCOPY("filename", "esc-sequence")
```

3.45.2. Description

BHCOPY produces screen dumps with dot matrix printers like EPSON or compatible ones in bit image mode.

Creating graphics on dot matrix printers is easy and fast. These hardcopy routines work in the standard mode of the Extended Basic II Plus, too.

The parameters are:

<i>filename</i>	Printer options Usual: "RS232.BA=9600.DA=8,CR" or "PIO.CR" The dip switches on the printer have to be set accordingly
<i>esc-sequence</i>	Printer adjustment e.g.: "K" or "L" for normal or double density in bit image mode

3.45.3. CAUTION

The *filename* extension .CR is essential for these routines to work correctly. The printer must have set the inverse signal AUTO FEED XT internally to ON.

Using serial interfaces (RS232) the extension .DA=8 must be added. The printer needs to be set to 8 data bits too.

esc-sequence depends on the printer. You need to consult the printer manual for the values to be used. BHCOPY sends at the beginning of every line CHR\$(27) (=ESC), then *esc-sequence* (max. 10 characters), then the characters CHR\$(0) and CHR\$(1) (for the subsequent 256 bytes). The 256 bytes corresponding to the hardcopy of one screen line are sent next, ending with CHR\$(13) (for CR). This respects the data transfer sequence of TI Line Printer (EPSON) using the bit image mode.

3.45.4. Format samples

```
CALL BHCOPY("RS232.BA=9600.DA=8.CR", "K")  
CALL BHCOPY("PIO.CR", "L")
```

3.45.5. Examples

```
100 REM SAVE DSK1.3455EPSON
110 REM ***EPSONCOPY***
120 CALL LINK("GRAFIC",0)
130 CALL LINK("WINDOW",3,8)
140 CALL LINK("CENTRE",64,60)
150 FOR PHI=0 TO 355 STEP 5
160 CALL LINK("SETTO",0,0)
170 CALL LINK("TURNT0",PHI)
180 CALL LINK("MOVE",50)
190 NEXT PHI
200 CALL LINK("WRITE",15,6,"BHCOPY")
210 POPT$="PIO.CR"
220 OPEN #1:POPT$,OUTPUT
230 PRINT #1:CHR$(27);"A";CHR$(8)
240 CLOSE #1
250 CALL BHCOPY(POPT$,"K")
260 STOP
```

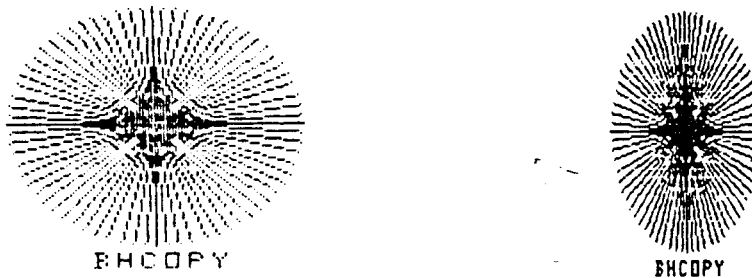


Fig. 3. BHCOPY

The reproduction of the radiating wreath on the printer is generated by BHCOPY in the proportion 1:1 in single density ("K"), but is deformed elliptically using double density ("L").

3.46. Hardcopy demonstration program

The following program example is called HCOPYDEMO. This program demonstrates how various graphic patterns are created.

This program is a useful aid studying the output of graphic to a matrix printer.

```
100 ! SAVE DSK1.3460HCDEMO
110 ! ***HCOPYDEMO***
120 ! By Apesoft 19850301
130 POPT$="PIO.CR" !3-D OPTION
140 ! Line feed reduction to 8 pixel rows
150 !
160 LF8$=CHR$(27)&"A"&CHR$(8)
170 OPEN #1:POPT$,OUTPUT
180 PRINT #1:LF8$
190 CLOSE #1
200 !
210 ! Pattern for "APESOFT" copyright sign
220 !
230 DATA 96,000001070F1F1F3F,97,3F3F7F7F7F7F7F7F,
    98,003FFFFFFFFFFFFFFF,99,E3C1808080C1E3FF
240 DATA 100,030F1F3F3F7F7F7F,101,7F7F7F7F7F7F7F7F,
    102,60787C7E7E7E7E7F7F,103,7F7F7E7E7E7C7860
250 DATA 104,7F7F7F7F7F7F7F7F
260 DATA 106,030F1F3F3F3F7F7F,107,7F7F3F3F3F1F0F03,
    108,60787C7E7E7E7E7F7F,109,00007E7E7E7C7860
270 DATA 110,030F1F3F3F3F7F7F,111,7F00003F3F1F0F03,
    112,E0F8FCFEFEFE8080,113,FFFFFFFFEFCF8E0
280 DATA 114,030F1F3F3F7F7F,115,7F7F7F3F3F1F0F03,
    116,60787C7E7E7E7E7F7F,117,7F7F7E7E7C7860
290 DATA 118,00000030F1F3F3F,119,0000001C3E7F7F7F,
    120,3F7F7F7F7F7F7F7F,121,7F7F7F7F7F7F7F7F
300 DATA 122,3E1C000000006060,123,6060606
310 DATA 124,0000007F7F7F7F7F0,125,000000000000006,
    126,7F7F7F7F7F7F7F7F,127,7F7F7F3F3F1F0F03
320 DATA 128,6060606060000000,129,001C3E7F7F7F3E1C
330 DATA 130,00FCFFFFFFFFFFFFFF,131,9F87838181808080,
    132,000080E0F0F8F8FC,133,FCFCFEFEFEFEFEFE
340 DATA 134,7F7F7F7F7F7F7F3F,135,3F3F1F1F0F070100,
    136,8080C0C0C0E0F0FC,137,FFFFFFFFFFFFFFF3F
350 DATA 138,8080808080808080,139,FFFFFFFFFC0FFFF,
    140,FEFEFEFEFEFEFCFC,141,FCF8F8F0E000FEFE
360 ! Transfer of Ape characters
370 CALL CLEAR
380 CALL SCREEN(2)
390 READ I,N$
400 CALL CHAR(I,N$)
410 IF I<141 THEN 390
420 PRINT TAB(9);"MICROCOMPUTER"::
```

TEXAS INSTRUMENTS HOME COMPUTER

```
430 PRINT TAB(7);" `b";CHR$(130);CHR$(132);"          vw|}"
440 PRINT TAB(7);"ac";CHR$(131);CHR$(133);"dfjlnprt{xz~";CHR$(128)
450 PRINT TAB(7);CHR$(134);CHR$(136);CHR$(138);CHR$(140);
    "egkmoqsuy{";CHR$(127);CHR$(129)
460 PRINT TAB(7);CHR$(135);CHR$(137);CHR$(139);CHR$(141);"h"
470 PRINT:TAB(11);"SOFTWARE"::::::::::::::::::
480 CALL SCREEN(4)
490 CALL BHCOPY(POPT$,"K")
500 CALL ALLSET
510 !
520 ! 3-D cube
530 !
540 CALL LINK("GRAFIC",0)
550 CALL LINK("CENTRE",1,-2)
560 CALL LINK("WINDOW",1,1)
570 CALL LINK("TURNTO",41,8103)
580 FOR Y=-30 TO -95 STEP -3
590 CALL LINK("SETTO",X,Y)
600 CALL LINK("MOVE",40)
610 NEXT Y
620 Y=-55
630 CALL LINK("TURNTO",90)
640 FOR X=32 TO 117 STEP 3
650 Y=Y+1
660 CALL LINK("SETTO",X,Y)
670 CALL LINK("MOVE",65)
680 NEXT X
690 CALL LINK("TURNTO",-18.4349)
700 Y=-30
710 FOR X=1 TO 33 STEP 3
720 CALL LINK("SETTO",X,Y)
730 CALL LINK("MOVE",92)
740 Y=Y-2.5
750 NEXT X
760 CALL BHCOPY(POPT$,"K")
770 !
780 ! Pyramid
790 !
800 CALL LINK("CLTBLE")
810 CALL LINK("CENTRE",64,1)
820 YG=-96
830 FOR XG=-63 TO -20 STEP 2
840 CALL LINK("SETTO",0,0)
850 CALL LINK("MOVETO",XG,YG)
860 YG=YG-1
870 NEXT XG
880 CALL LINK("SETTO",0,0)
890 CALL LINK("REMVTO",XG,YG)
900 FOR XG=XG+3 TO 46 STEP 3
910 CALL LINK("SETTO",0,0)
920 CALL LINK("MOVETO",XG,YG)
```

```
930 YG=YG+3
940 NEXT XG
950 CALL BHCOPY(POPT$, "K")
960 !
970 ! Cylinder
980 !
990 CALL LINK("CLTBLE")
1000 CALL LINK("SETBLE",12)
1010 CALL LINK("WINDOW",-1,4)
1020 CALL LINK("CENTRE",48,1)
1030 CALL LINK("TURNTO",0)
1040 FOR Y=-120 TO -40 STEP 3
1050 CALL LINK("ELLIPS",0,Y,48,24)
1060 NEXT Y
1070 FOR R=48 TO 2 STEP -2
1080 CALL LINK("ELLIPS",0,Y,R,R/2)
1090 NEXT R
1100 CALL BHCOPY(POPT$, "K")
1110 !
1120 ! Cyclone
1130 !
1140 PI2=8*ATN(1)
1150 CALL LINK("GRAFIC",1)
1160 CALL LINK("WINDOW",5,8)
1170 CALL LINK("CENTRE",1,40)
1180 CALL LINK("AXIS",8,116,0,4,0,36,40,4)
1190 X=7
1200 CALL LINK("SETTO",X,-20)
1210 FOR PHI=0 TO 3*PI2 STEP PI2/16
1220 X=X+2
1230 Y=20*(SIN(2*PHI)*COS(PHI)*2)
1240 CALL LINK("MOVETO",X,Y)
1250 NEXT PHI
1260 CALL LINK("DSPLAY",16,9,8,"AVERAGE")
1270 CALL LINK("DSPLAY",17,9,13,"VALUES")
1280 M$="TEMPERATURE"
1290 FOR Z=5 TO 15
1300 CALL LINK("DSPLAY",Z,7,1,SEG$(M$,Z-4,1))
1310 NEXT Z
1320 CALL LINK("WRITE",10,Z,"day-degrees")
1330 CALL BHCOPY(POPT$, "K")
1340 !
1350 ! Histograms
1360 !
1370 CALL LINK("GRAFIC",0)
1380 CALL LINK("SETBLE",24)
1390 CALL LINK("WINDOW",1,1)
1400 CALL LINK("HSTDIA",1,68,4,8,2,8,56,8,16,4,24,32,12,32,8)
1410 CALL LINK("HSTDIA",50,0,18,56,11)

1420 CALL BHCOPY(POPT$, "K")
```

TEXAS INSTRUMENTS HOME COMPUTER

```
1430 !
1440 ! Torus
1450 !
1460 CALL LINK("CLTBLE")
1470 CALL LINK("CENTRE",96,40)
1480 PIARC=4*ATN(1)
1490 FOR PHI=0 TO 2*PIARC STEP PIARC/40
1500 CALL LINK("CIRCLE",66*SIN(PHI),20*COS(PHI),17)
1510 NEXT PHI
1520 CALL BHCOPY(POPT$, "K")
1530 !
1540 ! Flowers
1550 !
1560 CALL LINK("GRAFIC",0)
1570 CALL LINK("WINDOW",1,4)
1580 CALL LINK("CENTRE",54,48)
1590 CALL LINK("SETTO",0,0)
1600 FOR PHI=0 TO 22.5 STEP 22.5
1610 CALL LINK("TURNTO",PHI)
1620 FOR X=1 TO 7
1630 FOR W=0 TO 15
1640 CALL LINK("MOVE",4)
1650 CALL LINK("TURN",W)
1660 NEXT W
1670 FOR W=16 TO 4 STEP -1
1680 CALL LINK("MOVE",4)
1690 CALL LINK("TURN",W)
1700 NEXT W
1710 CALL LINK("MOVETO",0,0)
1720 CALL LINK("TURN",6)
1730 NEXT X
1740 NEXT PHI
1750 FOR R=1 TO 12
1760 CALL LINK("CIRCLE",0,0,R)
1770 NEXT R
1780 CALL BHCOPY(POPT$, "K")
1790 !
1800 LF$=CHR$(10) !LINE FEED
1810 OPEN #1:POPT$,OUTPUT
1820 PRINT #1:CHR$(27);CHR$(2) !RESET TO STANDARD LINE FEED
1830 PRINT #1:LF$;LF$;LF$;LF$:" H I G H R E S O L U T I O N":LF$;LF$
1840 PRINT #1:LF$;"          G R A P H I C";LF$;LF$
1850 CLOSE #1
1860 CALL LINK("BYEBYE")
1870 END
```

4. Appendix

4.1. Alphabetic quick reference

The order of statements disregards any preceding CALLs or CALL LINKs.

CALL LINK("ACCEPT", *r*, *c*, *size*, *var*\$)

accepts *size* characters of the string *var*\$ at position (*r*,*c*).

CALL ALLSET

resets the ASCII characters 32 to 126 to their standard definitions.

CALL APESOFT

transfers the high resolution graphic routines into the RAM expansion. This command must be entered before loading of BASIC programs.

CALL LINK("ARCUS", *x*, *y*, *r*, *phi*, *dphi*[, *x1*, *y1*, *r1*, *phil*, *dphil*, ...])

draws arcs with center point *x*,*y*, radius *r*. starting angle *phi*, and arc angle *dphi*.

CALL LINK("AXIS", *x*, *lenxl*, *lenxr*, *deltax*, *y*, *lenu*, *lency*, *deltay*)

draws a system of coordinates with the center point *x*,*y*. the left X-semi-axis *lenxl*, the right X-semi-axis *lenxr*, the pitch of the X-grid *deltax*, the upper Y-semi-axis *lenu*, the bottom Y-semi-axis *lency* and the pitch of the Y-grid *deltay*.

CALL BHCOPY("filename", "esc-sequence")

generates screen copies on EPSON or compatible printers in bit image mode. *filename* contains the interface options and *esc-sequence* contains the printer adjustment.

CALL BYE

erases the loaded programs and data and calls the master screen. The BASIC Mode is unchanged.

CALL LINK("BYEBYE")

cancel *Graphic Mode*, loads the standard character set and reestablishes Extended Basic II Plus *Standard Mode*.

TEXAS INSTRUMENTS HOME COMPUTER

CALL LINK("CENTRE" , *x* , *y*)

defines the system of user defined coordinates with the (0,0)-point the position (*x*,*y*) of the graphic table.

CALL LINK("CIRCLE" , *x* , *v* , *r* [, *x1* , *v1* , *r1* , . . .])

draws a circle with the center point *x*,*y* and the radius *r*.

CALL CLRRAPE

initializes Extended BASIC just like after selection of MECHATRONIC EXTENDED BASIC from the TI selection screen. It may only be entered after execution of CALL APESOFT otherwise it will cause a syntax error.

CALL LINK("CLARCS" , *x* , *y* , *r* , *phi* , *dphi* [, *x1* , *y1* , *r1* , *phil* , *dphil* , . . .])

erases the arc pixels.

CALL LINK("CLCRCL" , *x* , *y* , *r* [, *x1* , *y1* , *r1* , . . .])

erases the circles.

CALL LINK("CLLIPS" , *x* , *y* , *a* , *b* [, *x1* , *y1* , *a1* , *b1* , . . .])

erases the ellipses.

CALL LINK("CLRECT" , *a* , *b* [, *a1* , *b1* , . . .])

erases the rectangles

CALL LINK("CLSCRN")

clears the screen. The graphic in the table together with all other internal parameters remain.

CALL LINK("CLTBLE")

clears the graphic table and thus the graphic.

CALL LINK("CRCRIA" , *x* , *y* , *rad* , *phi* , *dphi* [, *x1* , *y1* , *rad1* , *phil* , *dphil* , . . .])

draws a circular diagram with the coordinates of the circular segment center point X,Y, radius RAD, starting angle PHI, and final angle of the circular segment DPH.

CALL LINK("DSPLAY" , *r* , *c* , *size* , *var*\$)

sets *size* characters of the string *var*\$ at position (*r*,*c*).

CALL LINK("ELLIPS" , *x* , *y* , *a* , *b* [, *x1* , *y1* , *a1* , *b1* , ...])

draws ellipses with axis center point *x*,*y*, major semi-axis *a*, minor semi-axis *b* and inclination *phi* to the major semi-axis.

CALL FIND("get-string" , "string array" () , *return variable*)

looks in a one-dimensional *string array* for *get-string*. *return variable* carries the number of the wanted element. If *get-string* is not found, *return variable* is assigned the value -1.

CALL GLOAD("filename")

loads a graphic called *filename* from diskette into VDP-RAM.

CALL GPEEK(*address* , *numeric variable list*)

reads the contents of sequential GROM addresses.

CALL LINK("GRAFIC" , *mode*)

sets the graphic mode, initializes all the computer registers, and defines a graphic table (max. 128 vertical and 120 horizontal lines) depending on *mode*: graphic or text mode).

CALL GSAVE("filename")

saves the colors of the graphic, the position of the graphic window, and the graphic parameters in memory image format on diskette.

CALL LINK("HSTDIA" , *x* , *y* , *width* , *height* , *depth*)

draws a block diagram with the coordinates of the left bottom corner of the block *x*,*y*, with width *width*, height *height*, and depth *depth*.

CALL LINK("IFSET" , *x* , *y* , *var* [, *x1* , *y1* , *var1* , ...])

checks if a pixel with coordinates *x*,*y* is set or not and returns the result to the variable *var*.

CALL LINK("INVERT" , *x* , *y* , *dx* , *dy*)

inverts sections of the graphic. *x*,*y* is the pixel position of the upper left corner, *dx* the pixel column position, and *dy* the pixel row position of the inverted graphic section.

TEXAS INSTRUMENTS HOME COMPUTER

CALL MLOAD("*filename*" ,*mode*)

loads saved CPU RAM contents back into CPU memory. Extended BASIC II Plus does not require *mode*. If *mode* is positive, an auto-start assembly language program file execution will be performed.

CALL MOVE (*mode* , *start address* , *target address* , *bytes*)

moves the contents of memory blocks with length *bytes* depending on *mode* (1-4) between VDP RAM and CPU RAM or within VDP RAM or CPU RAM.

CALL LINK("MOVE" , *dist*)

draws a line of length *dist*, starting from the present position *x,y* of the cursor with the present internal angle *phi*.

CALL LINK("MOVETO" , *x* , *y* [, *x1* , *y1* , . . .])

draws a line from the present internal position of the cursor to the next one by the parameter pair *x,y* defined position.

CALL MSAVE("*filename*" , *start address* , *bytes*)

saves memory blocks of CPU RAM with length *bytes* to an external device in program image format.

CALL NEW

erases the BASIC program and data in RAM and prepares the computer for receiving of new BASIC programs.

CALL QUITOF

deactivates the QUIT function (**FCTN =**).

CALL QUITON

reactivates the QUIT function.

CALL LINK("RECT" , *a* , *b* [, *a1* , *b1* , . . .])

draws rectangles in the order *a -> b -> a -> b* using the actual cursor parameters.

CALL LINK("REMOVE" , *dist*)

deletes all pixels from position *x,y* to the *dist* new position of the cursor.

CALL LINK("REMVTO",*x,y[,x1,y1,...]*)

deletes the lines.

CALL LINK("RESET",*x,y[,x1,y1,...]*)

erases pixels with the coordinates *x,y*.

CALL RESTORE(*numeric variable*)

prepares the computer to process the next DATA Statement using the line number in *numeric variable*.

CALL SCREENOF

switches the screen off.

CALL SCREENON

reactivates the screen.

CALL LINK("SETBLE",*width*)

dimensions the graphic table.

CALL LINK("SETCOL",*foreground color,background color*)

changes *foreground color* and *background color* simultaneously for the entire graphic.

CALL LINK("SETCOL",*n,fg,bg[,n1,fg1,bg1,...]*)

defines foreground color *fg* and background color *bg* for the character set specified by *n*.

CALL LINK("SETTO",*x,y[,x1,y1,...]*)

sets pixels with the coordinates *x* and *y*.

CALL LINK("SHIFT",*deltax,deltay*)

transforms a graphic linear by the values *deltax* in column direction and *deltay* in row direction.

TEXAS INSTRUMENTS HOME COMPUTER

CALL SPROF

stops the movement of all sprites at once.

CALL SPRON

restarts all stopped sprites.

CALL LINK("TABLE", *r*, *c*, *xmax*, *ymax*, *bytes*)

returns the present parameters of the graphic table to the variable list. *r* contains the number of rows, *c* the number of columns, *xmax* the maximum number of pixel columns, *ymax* the maximum number of pixel rows of the table, and *bytes* to the number of available bytes.

CALL LINK("TURN", *phi*)

adds to the present internal angle of the cursor the angle *phi* in decimal degrees. *phi* turns clockwise.

CALL LINK("TURNT0", *phi*)

sets the internal angle imperatively to *phi* (degrees).

CALL LINK("VALUES", *x*, *y*, *phi*, *fg*, *bg*)

returns the present internal parameters to the variable list. *x* is the cursor column, *y* the cursor row, *phi* the cursor angle, *fg* the foreground color, and *bg* the background color.

CALL VPEEK(*address*, *numeric variable list*)

reads the contents of sequential addresses in VDP RAM, starting with *address*, and returns them in *numeric variable list*.

CALL VP0KE(*address*, *numeric data*)

writes *numeric data* into sequential addresses in VDP RAM, starting with *address*.

CALL WAIT(*duration*)

delays program execution by *duration* * 1/60 seconds.

CALL LINK("WINDOW", *row*, *column*)

sets the entire graphic table — 16 x 8 columns, 15 x 8 rows (or 11 x 8 rows in text mode) at position *column* (1-32) and *row* (1-24).

CALL LINK("WINDOW" , *r* , *c* , *x* , *y* , *rc* , *cc*)

transmits sections of the graphic table to the screen: *r* is the row, *c* is the column, *x* and *y* are the upper left corner point of the graphic table, *rc* is the number of graphic table characters in row direction and *cc* the number of graphic table characters in column direction.

CALL LINK("WRITE" , *r* , *c* , *string*[, *r1* , *c1* , *string1* , ...])

enables the mixing of graphics and text in the graphic table and writes *string* at position row (*r*) and column (*c*) of the graphic table.

4.2. Reference list for Mechatronic Extended Basic II Plus

The following abbreviations are used:

- (I) Only allowed in Immediate mode
- (C) CALL *subroutine*
- (L) CALL LINK *assembly routine*

Reference Name	I	C	L	Mode		See Reference Manual:	
				Standard	Graphic	TI Extended Basic	Extended Basic II Plus
ABS				X	X	X	
ACCEPT				X		X	
ACCEPT			L		X		X
ALLSET		C		X			X
APESOFT	I	C		X			X
ARCUS			L		X		X
ASC				X	X	X	
ATN				X	X	X	
AXIS			L		X		X
BHCOPY		C		X	X		X
BREAK				X	X	X	
BYE	I			X		X	
BYE		C		X	X		X
BYEBYE			L	X	X		X
CALL				X	X	X	
CENTRE			L		X		X
CHAR		C		X	X	X	
CHARPAT		C		X	X	X	

Mechatronic Extended Basic II Plus

Reference Name	I	C	L	Mode		See Reference Manual:	
				Standard	Graphic	TI Extended Basic	Extended Basic II Plus
CHARSET		C		X		X	
CHR\$				X	X	X	
CLARCS			L		X		X
CLEAR		C		X		X	
CLLIPS			L		X		X
CLOSE				X	X	X	
CLRAPE	I	C		X	X		X
CLSCRN			L		X		X
CLTBLE			L		X		X
CONIC		C		X		X	
COLOR		C		X		X	
CON	I			X		X	
CONTINUE	I			X		X	
COS				X	X	X	
CRCDIA			L		X		X
DATA				X	X	X	
DEF				X	X	X	
DELETE				X	X	X	
DELSPRITE		C		X		X	
DIM				X	X	X	
DISPLAY				X		X	
DISPLAY USING				X		X	
DISTANCE		C		X		X	

TEXAS INSTRUMENTS
HOME COMPUTER

Reference Name	I	C	L	Mode		See Reference Manual:	
				Standard	Graphic	TI Extended Basic	Extended Basic II Plus
DSPLAY			L		X		X
ELLIPS			L		X		X
END				X	X	X	
EOF				X	X	X	
ERR		C		X	X	X	
EXP				X	X	X	
FILES		C		X	X	X	
FIND		C		X	X		X
FOR TO STEP				X	X	X	
GCHAR		C		X	X	X	
GLOAD		C			X		X
GOSUB				X	X	X	
GOTO				X	X	X	
GRAFIC			L	X	X		X
GPEEK		C		X	X		X
GSAVE		C			X		X
HCHAR		C		X		X	
HSTDIA			L		X		X
IFSET			L		X		X
IF-THEN-ELSE				X	X	X	
IMAGE				X	X	X	
INIT		C		X		X	
INPUT				X		X	

Mechatronic Extended Basic II Plus

Reference Name	I	C	L	Mode		See Reference Manual:	
				Standard	Graphic	TI Extended Basic	Extended Basic II Plus
INT				X	X	X	
INVERT			L		X		X
JOYST		C		X	X	X	
KEY		C		X	X	X	
LEN				X	X	X	
LET				X	X	X	
LINK		C		X	X	X	
LINPUT				X		X	
LIST	I			X		X	
LOAD		C		X	X	X	
LOCATE		C		X		X	
LOG				X	X	X	
MAGNIFY		C		X		X	
MAX				X	X	X	
MERGE	I			X		X	
MIN				X	X	X	
MLOAD		C		X	X		X
MOTION		C		X		X	
MOVE		C		X	X		X
MOVE			L		X		X
MSAVE		C		X	X		X
NEW	I			X		X	
NEW		C		X	X		X

TEXAS INSTRUMENTS
HOME COMPUTER

Reference Name	I	C	L	Mode		See Reference Manual:	
				Standard	Graphic	TI Extended Basic	Extended Basic II Plus
NEXT				X	X	X	
NUM	I			X		X	
NUMBER	I			X		X	
OLD	I			X		X	
ON BREAK				X	X	X	
ON ERROR				X	X	X	
ON GOSUB				X	X	X	
ON GOTO				X	X	X	
ON WARNING				X	X	X	
OPEN				X	X	X	
OPTION BASE				X	X	X	
PATTERN		C		X		X	
PEEK		C		X	X	X	
PI				X	X	X	
POS				X	X	X	
POSITION		C		X		X	
PRINT				X		X	
PRINT USING				X		X	
QUITON			L	X	X		X
QUITOF			L	X	X		X
RANDOMIZE				X	X	X	
READ				X	X	X	
REC				X	X	X	

Mechatronic Extended Basic II Plus

Reference Name	I	C	L	Mode		See Reference Manual:	
				Standard	Graphic	TI Extended Basic	Extended Basic II Plus
REM				X	X	X	
RES	I			X		X	
RESEQUENCE	I			X		X	
RESET			L		X		X
RESTORE				X	X	X	
RESTORE		C		X	X		X
RETURN				X	X	X	
RND				X	X	X	
RPT\$				X	X	X	
RUN				X	X	X	
SAVE	I			X		X	
SAY		C		X	X	X	
SCREEN		C		X	X	X	
SCREENON		C		X	X		X
SCREENOF		C		X	X		X
SEG\$				X	X	X	
SETBLE			L		X		X
SETCOL			L		X		X
SETTO			L		X		X
SGN				X	X	X	
SIN				X	X	X	
SIZE	I			X		X	
SOUND		C		X		X	

TEXAS INSTRUMENTS
HOME COMPUTER

Reference Name	I	C	L	Mode		See Reference Manual:	
				Standard	Graphic	TI Extended Basic	Extended Basic II Plus
SPGET		C		X	X	X	
SPRITE		C		X		X	
SPRON		C		X			X
SPROF		C		X			X
SQR				X	X	X	
STOP				X	X	X	
STR\$				X	X	X	
SUB				X	X	X	
SUBEND				X	X	X	
SUBEXIT				X	X	X	
TAB				X		X	
TABLE			L		X		X
TAN				X	X	X	
TRACE				X		X	
UNBREAK				X	X	X	
UNTRACE				X		X	
VAL				X	X	X	
VALUES			L		X		X
VCHAR		C		X		X	
VERSION		C		X	X	X	
VPEEK		C		X	X		X
VPOKE		C		X	X		X
WAIT		C		X	X		X

Mechatronic Extended Basic II Plus

<i>Reference Name</i>	<i>I</i>	<i>C</i>	<i>L</i>	<i>Mode</i>		<i>See Reference Manual:</i>	
				<i>Standard</i>	<i>Graphic</i>	<i>TI Extended Basic</i>	<i>Extended Basic II Plus</i>
WINDOW			L		X		X
WRITE			L		X		X

4.3. Memory mapping

Commands in the extended statement set of Mechatronic Extended Basic II Plus like CALL VPEEK, VPoke, MOVE, MSAVE, MLOAD require detailed knowledge of the system organization, if their full power is to be used. The following is a short survey of the memory mapping of the TI-99/4A, which may give some guidance. It is beyond the scope of this Extended Basic reference manual to provide full details of the TI operating system or the TI BASIC interpreter. More detailed information can be found in the Editor/Assembler manual.

4.3.1. TI-99/4A memory map

The TMS9900 microprocessor has an address space of 64K bytes. In the Home Computer, some of this address space contains RAM and some contains ROM. In addition, some addresses are used for access to special devices, such as sound and speech, and other areas of memory, such as VDP RAM and GROMs. These are called memory-mapped addresses.

The following memory map shows directly addressable memory:

<i>Hex</i>	<i>CPU memory use — general case</i>	<i>Decimal</i>
>0000	Console ROM. Two 4K chips	0000
>2000	Low Memory Expansion (8K bytes)	8192
>4000	Peripheral ROMs (mapped) up to 8K bytes for Device Service Routine	16384
>6000	Application ROMs in Command Module	24576
>8000	Memory-mapped devices for VDP, GROM, Sound, and Speech	32768
>A000	High Memory Expansion (24K bytes)	40960

4.4. ROMs and GROMs

All ROM (Read Only Memory) addresses are directly accessible by an assembly language program. Two 4K byte console ROMs are located at addresses >0000 through >1FFF. They contain the operating system, the GPL interpreter, and parts of the TI BASIC interpreter.

4.4.1. GROMs

A GROM (Graphics Read Only Memory) is another type of ROM. It is designed to contain GPL (Graphics Programming Language) programs which are executed by the GPL interpreter in the console. GPL is commonly used in applications software and can only be executed through a GROM.

4.5. VDP RAM

The VDP (Video Display Processor) RAM, located in the console, is chiefly used for common video functions, such as storing the screen image, character pattern table, color table, etc.

When Extended BASIC is in use, VDP RAM also contains the BASIC program, the program symbol table, the value stack, and the string space. The VDP RAM is also used as storage space by application programs. Part of VDP RAM is used as a data buffer. Another part of VDP RAM functions as a PAB (Peripheral Access Block) to pass information from a file to an appropriate DSR (Device Service Routine). Assembly language programs cannot be executed from VDP RAM.

VDP RAM is a memory-mapped area of 16K (16384 or >4000) bytes numbered >0000 through >3FFF. VDP RAM addresses are automatically incremented, so only one address in CPU RAM is required to read or write a specific block of data

TEXAS INSTRUMENTS
HOME COMPUTER

<i>Hex</i>	<i>VDP memory use by Extended Basic</i>	<i>Decimal</i>
>0000	Screen	0000
>02FF		0767
>0300	Sprite attribute list	0768
>0370		0880
>0371	Basic temporaries	0881
>03EF		1007
>03F0	Character tables	1008
>077F		1919
>0780	Sprite motion table	1920
>07FF		2047
>080F	Color table	2063
>081F		2079
>0820	Basic crunch buffer	2080
>0957		2391
>0958	Value Stack String Space Symbol Table Line Number Table Crunched Program PAB	2392
>3FFF		16383

4.5.1. Notes on VDP RAM use

4.5.1.1. Screen Images

The 768 characters (32 columns x 24 rows) of the screen are arranged line by line by at addresses 0 through 767 starting in the upper left corner. Every character is represented by one byte. This byte has an offset of 96 (>60) added to the value of the ASCII Code of the displayed character. (ASCII value + 96).

4.5.1.2. Sprite Attributes

This area contains the data for sprites #1 through 28. Each sprite uses 4 bytes in the following order:

1. Y-Position (Pixel row 1 = 255)
(Pixel row 2 = 0)
(Pixel row 3 = 1) etc
2. X-Position (Pixel column 1 = 0)
(Pixel column 2 = 1) etc
3. Character Code (Offset of ASCII codes of sprites character)
4. Color Code (BASIC color code minus 1)

4.5.1.3. Character Patterns

The patterns of ASCII codes 30 through 143 are defined in 8 consecutive bytes. The character definition corresponds to that of BASIC with one exception: each byte in hexadecimal notation has to be converted to decimal notation. (1 byte = 2 hex digits. >00 - >FF corresponds to dec 00-255).

Example:

```
CALL CHAR(32,007C7C7C7C7C7C7C)
CALL VPOKE(1024,0,124,124,124,124,124,124,124)
```

If sprite motion is not used, ASCII Codes 144-159 may be used.

4.5.1.4. Sprite motion

This area contains the velocities of sprites #1 through #28. Each sprite is represented by 4 bytes in the following order:

1. Vertical speed
2. Horizontal speed
3. Used by operating system
4. Used by operating system

4.5.1.5. Color table

Information about both foreground and background color of the character sets 0 through 14 are stored in one byte. The value is the result of foreground color code minus 1 times 16 plus background color code minus 1. e.g. entering CALL VPOKE(2063,96) results a dark red cursor during program execution.

4.5.1.6. Crunch Buffer

Addresses 2240 through 2391 can contain a maximum 151 characters from the last keyboard entry which may be recalled by **REDO (FCTN 8)**. The values have the BASIC offset of 96 added.

4.5.1.7. Basic Programs

User data such as crunched BASIC program lines and the corresponding line numbers, symbol tables, string space, etc, do not have fixed addresses. They depend on the system configuration and the program itself.

CAUTION

Indiscriminate use of the statements VPOKE or MOVE can cause a system crash and loss of stored program and data. Therefore, before experimenting, it is strongly recommended that you back up the current program.

Statement of File Origin

This file was created for users of PC99, a TI-99/4A emulator running on an IBM PC.

Reproduced with permission of the publisher Copyright (1983) Mechatronic Elektronische Bauelemente GmbH & Co, Germany. This file was created by scanning the manual originally supplied with the Mechatronic Extended Basic II Plus product. Although Mechatronic has granted permission for this, Mechatronic is in no way responsible for any errors, omissions or changes introduced by this process. In the case of a dispute, the user of this file is referred to the original manual.

The text and graphic contents of this file are based on the original Mechatronic manual, which was translated from German to English. This file contains extensive revisions and corrections to the original Mechatronic manual. While every effort was made to ensure that the changes remain faithful to the original Mechatronic manual, CaDD Electronics can assume no responsibility for any errors introduced during scanning, editing, or conversion.

If you find an error, we will attempt to correct it and provide you with an updated file. You can contact us at:

**CaDD Electronics
45 Centerville Drive
Salem, NH 03079-2674**

Version 970712

CaDD Electronics would like to acknowledge the help and advice given in the preparation of this manual by Carsten Ziepke, Westring 268, D-24116 Kiel, Germany. Mr. Ziepke was instrumental in persuading Mechatronic to release this product as shareware for use with PC99. If you are using Mechatronic Extended Basic II Plus with PC99, it is suggested that a donation of \$5.00 (or approximate equivalent in other currencies) be sent to:

**Mechatronic Elektronische Bauelemente GmbH & Co
D-71065 Sindelfingen
Germany**

Because of the difficulties and expense of sending small dollar amounts overseas, it is probably easiest to simply enclose a \$5 bill (or approximate equivalent in other currencies) in a letter to Mechatronic thanking them for their generous offer.

We would also like to thank Charles Good for keying in the first pass of most of the programs in this manual.