User Manual

Turbo Vision 4 BASIC

By
Károly Nagy
(CHARLEMAGNE)
2009-2014.

Turbo Vision 4 BASIC
User Manual

By
Károly Nagy
(CHARLEMAGNE)

Turbo Vision 4 BASIC

User Manual

By
Károly Nagy
(CHARLEMAGNE)
Hungary, Pálmonostora

CHARLEMAGNE, 2009-2014.

Contents

Chapter 1

Introduction

The whole story begins on May 1, 1964 at 4 am when John Kemeny and John McGeachie ran the first BASIC[1] program: **PRINT 2 + 2** at Dartmouth[2] College.

An another gripping story tells about Jack Tramiel who (after surviving the Holocaust) set up his own typewriter repair business in Bronx while working as a cab driver at night. The company went commercial in 1962, calling itself CBM[3]. In 1977 (when I was born by the way :-)), CBM started producing PET[4] computers.

It is hard to develope a graphic software which can handle a wide range of video hardware. SGI[5] released OpenGL[6] in 1992. It is not so easy to handle a lot of peripheral devices, too. Well, Microsoft[7] released DirectX[8] in 1995. In the end, I needed lcc-win32[9] (changed to VC2010) to compile my sources. What's more, LaTeX[10] helped me to write down this manual.

Thanks a lot to the mentioned people and techniques above (and below) for me to create Turbo Vision 4 BASIC. TV4B is an own developed BASIC interpreter with several individual capabilities.

[1] **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode was designed in 1963. ...
[2] ... by John George Kemeny and Thomas Eugene Kurtz in New Hampshire, USA.
[3] **C**ommodore **B**usiness **M**achines was led by Jack Tramiel and C. Powell Morgan
[4] **P**ersonal **E**lectronic **T**ransactor was a home/personal computer
[5] **S**ilicon **G**raphics, **I**nc. founded in 1981 by Jim Clark and Abbey Silverstone
[6] **Open G**raphics **L**ibrary is a cross-language for easy-to-use 2D and 3D graphics
[7] Microsoft founded in 1975 by Bill Gates and Paul Allen
[8] **DirectX** is a collection of application programming interfaces such as D3D, DD, DS, DI
[9] Logiciels/Informatique lcc-win32 is a free C compiler by Jacob Navia
[10] LaTeX is a document markup language, written in 1984 by Leslie Lamport

1.1 The typewriter

If we make something clear or like to teach, the best way is that we show the very beginning. This notion led me to create a simple environment like a ”typewriter” to tell the computer what we want.

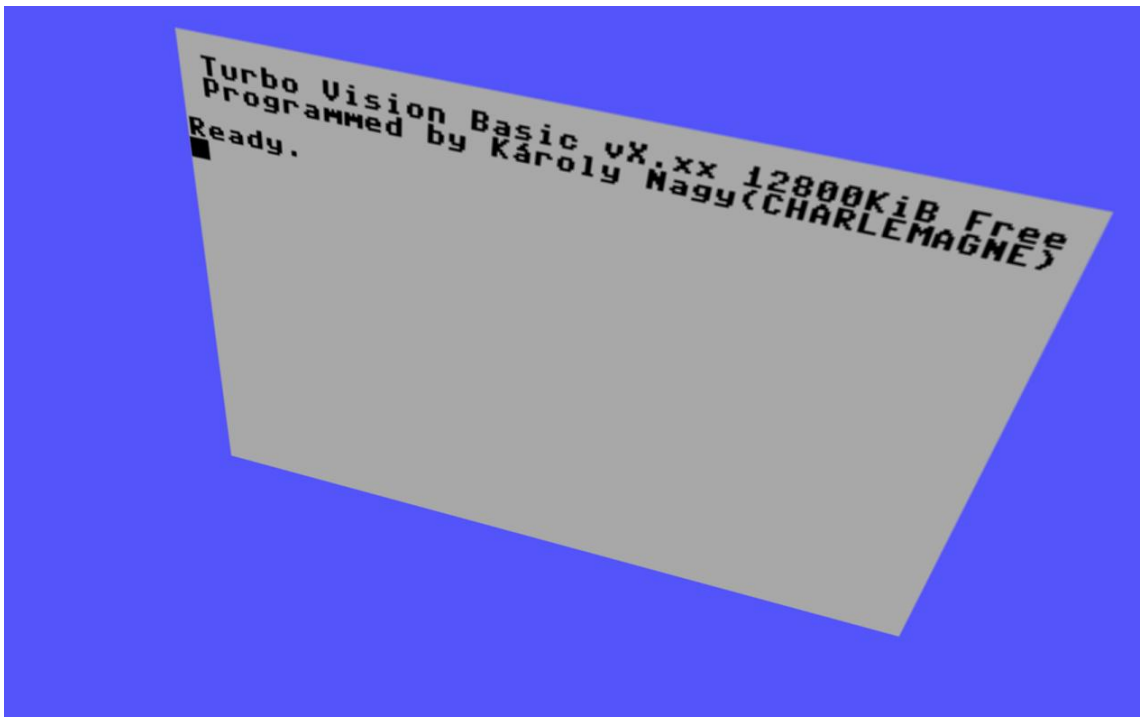

Figure 1.1: Base screen

The base screen consists of a work area and a frame. It is really an OpenGL surface which resizes itself to the current video resolution. You are able to move it with the following keys:

Keys	Action
(right)SHIFT+Up/Down	Rotate over X axis
(right)SHIFT+Left/Right	Rotate over Y axis
(right)CTRL+Left/Right	Rotate over Z axis
(right)CTRL+Up/Down	Zoom in/out
(right)SHIFT/CTRL+HOME	Back to the default

Table 1.1: Base keys

1.1.1 The work area and the frame

The work area is where you can type a text. This is an 512x256 POT texture in order to speed up the application! It has got 320 columns and 200 rows of pixels (320x200 part of the main texture updated). The frame is all around the work area.

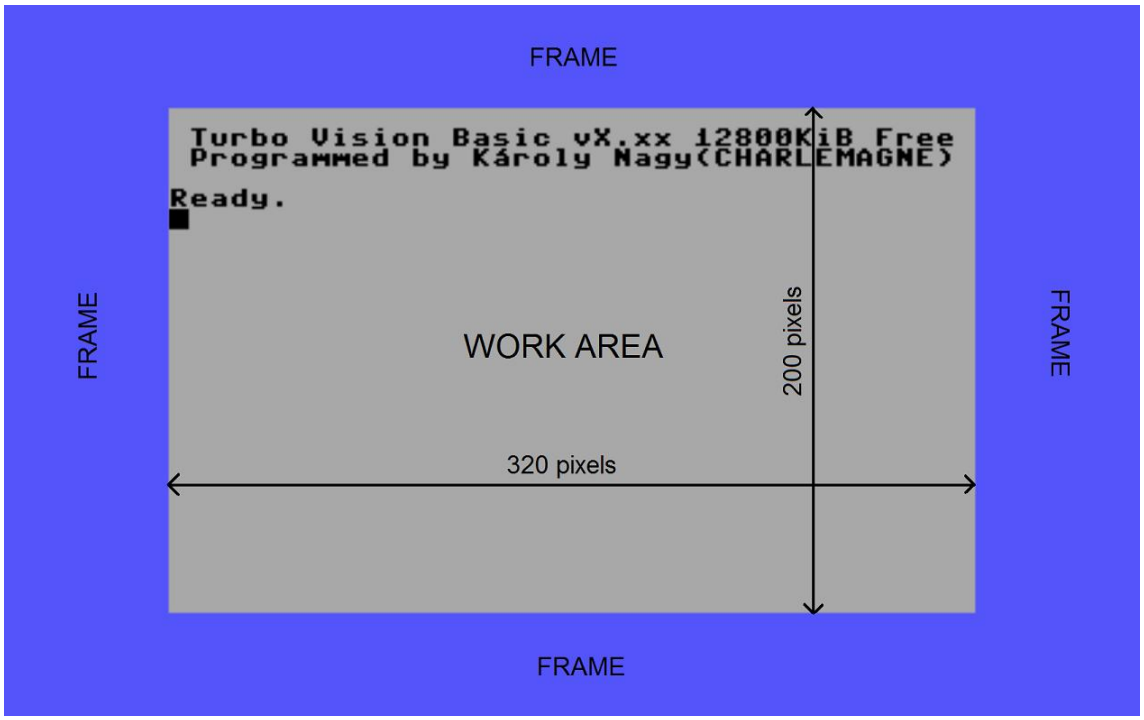

Figure 1.2: The work area and the frame

The default color is cyan (07) for the work area and lightblue (09) for the frame (black (00) for text). You can modify them of couse at [0]:0085[11] (ForeGround), [0]:0086 (BackGround) in text mode and at [0]:E87E (ForeGround), [0]:E87F (BackGround) in graphic mode. The frame color is the same in both case at [0]:E87D.

Why do you get involved in memory map? Would you like to be an advanced user? Okay! In turn I show you an easy-way to set colors instead of memory map: COLOR[12]0,4 for setting red background, COLOR1,14 for setting yellow foreground, COLOR4,15 for setting white frame.

[11] **[d]:hhhh** is a reference to the memory map where **d** is a decimal number for segment and **hhhh** is a hexadecimal number for offset

[12] **COLORw,c** is a BASIC instruction to set colors (w=0:BackGround; w=1:ForeGround; w=4:Frame; c=0..15: a color)

1.1.2 Editor

There is a cursor blinking on the work area. This filled rectangle enables you to see where the next character (what you type right now) will be shown up. There are 40 columns and 25 rows of characters on the text screen[13].

You can move the cursor with arrow-keys: up, down, left, right. When the cursor reaches the right border of the work area, it moves the beginning of the next line. It is also true in the backwards, only one difference is that the cursor moves the end of the previous line. A very interesting event occures at the right-bottom of the screen, the content of the work area moves up by one line. This happens when the cursor is in the last line of the work area and you press down-arrow-key (or ENTER).

The editor divides the work area into 25 parts, calling these parts sections. A section is a line at the start which extends automatically.

Type a text

If you press an alphabetic key on the keyboard, a lower case letter is going to be shown on the screen at the cursor position. You can also use Hungarian accentuated letters[14]. After showing up a character, the cursor moves right. You can write capital letters with SHIFT+the desired alphabetic key.

Moreover you are able to write special characters with (left)ALT+d where d is a number entering on the numeric keypad (right side of the keyboard).

Insert mode

The editor has got two typing mode. The one is overwrite-mode, the another is insert-mode (default is overwrite). You can switch between the two modes with INSERT-key[15].

After typing a text in overwrite-mode, you realize that each typed character write over the characters on the screen. But then typing in insert-mode, characters are inserted into the others on the screen pushing them right.

In that case the insert mode is on, the screen scrolling not only up with its content, but down when the cursor is in the first line and pressing the up-arrow-key.

[13] [0]:0C00-[0]:0FE7 is the area of the text screen (40x25 character)
[14] Árvíztűrő tükörfúrógép
[15] [0]:07EA is INSERT mode flag (0: insert off; 255: insert on)

Modify a text

Sometimes (or usually?) we make mistakes in the text and would like to alter it. You can delete the last typed character (on the left of the cursor) with BACKSPACE and the character on the right with DELETE.

ENTER

It enables you to run a command or add a new line to your BASIC or ASSEMBLY program. After pressing it, the TV4B tries to interpret the command you entered into the current section.

1.1.3 Some examples

Now you were getting the appropriate knowledge to write simple commands into the typewriter. Try to color the background, text and frame what you read in "The work area and the frame" section before.

Color index	Color name
0	Black
1	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Brown
7	Light Gray
8	Dark Gray
9	Light Blue
10	Light Green
11	Light Cyan
12	Light Red
13	Light Magenta
14	Yellow
15	White

Table 1.2: the first 16 colors

1.2 Control Keys

Control keys enable you to work easily in the typewriter.

Keys	**Action**
Arrows	Move the cursor
Alphabetic and numeric keys	the same
SHIFT/AltGr	Capital letter / signs
(left) CTRL+c	Graphical characters I. (c=a...z)
(left/right) CTRL+n	Color characters (n=1...8)
(left/right) CTRL+n	Reverse and Blink (n=0,9)
(left) ALT+c	Graphical characters II. (c=a...z)
(left) ALT+d	Special ASCII characters (d=0...255)
ENTER	Interpret the current section
BACKSPACE	Delete a character on the left
DELETE	Delete a character on the right
INSERT	Insert-mode on/off
HOME	Jump to the first character of the line
END	Jump to the last character of the line
F9	Mouse on/off
F10	Switch widescreen
F11	Reset TV4B
F12	Quit from TV4B
ESCAPE	Interrupt a running program

Table 1.3: Control keys

1.3 Graphical characters I.

You are able to draw shapes on the text screen with combinations of keys. The first set of graphical characters are available with LEFT CONTROL.

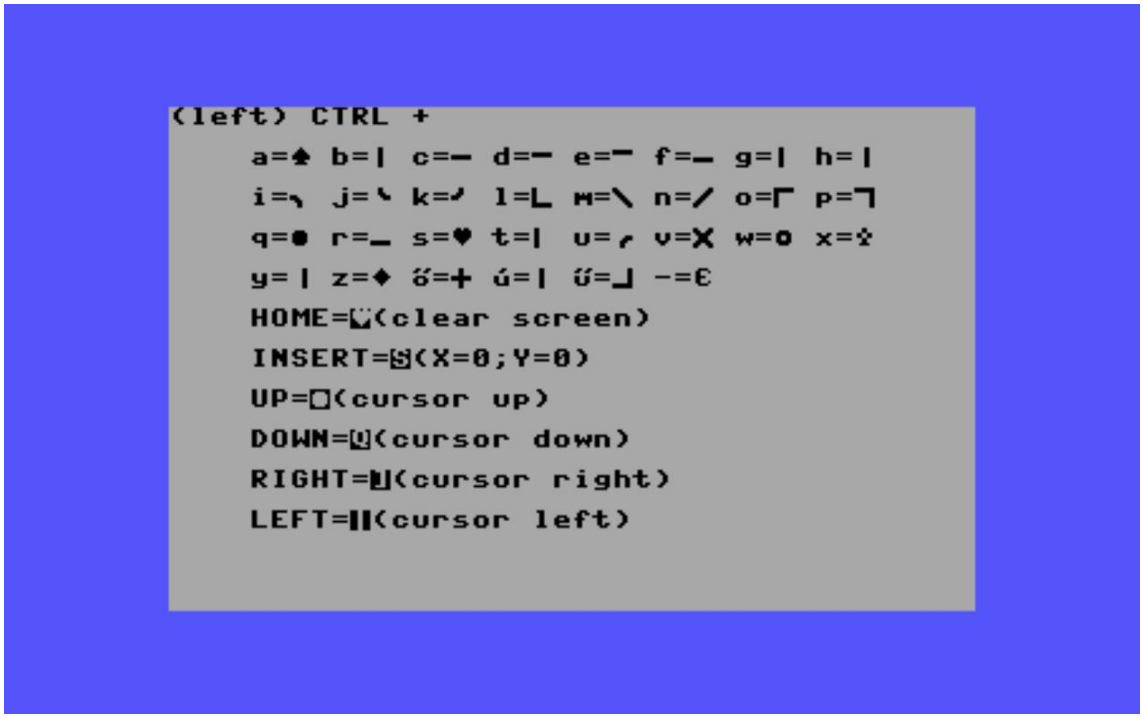

Figure 1.3: Graphical characters I.

eg.#1.:
Press 's' while pressing (left) CTRL –> a "heart"-shape will appear on the screen
eg.#2.:
In that case of writing "cursor down"-shape on the screen, the cursor is going to move down by how many times you write this character.

1.4 Graphical characters II.

You are able to draw shapes on the text screen with combinations of keys. The second set of graphical characters are available with LEFT ALT.

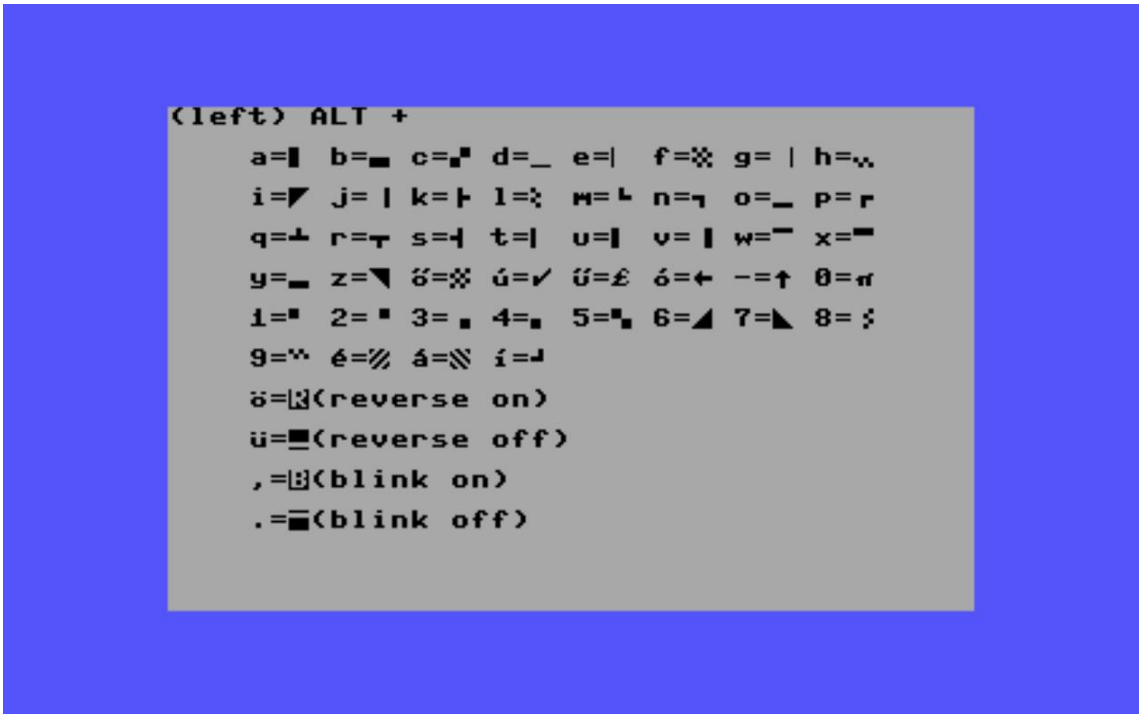

Figure 1.4: Graphical characters II.

eg.#1.:
Press '-' while pressing (left) ALT –> an "up arrow"-shape will appear on the screen. (2"up arrow"$5 = 2^5$)
eg.#2.:
In that case of writing "blink on"-shape on the screen, the text is going to blink (what you type or write).

1.5 Color characters

You are able to set the text color with special characters.

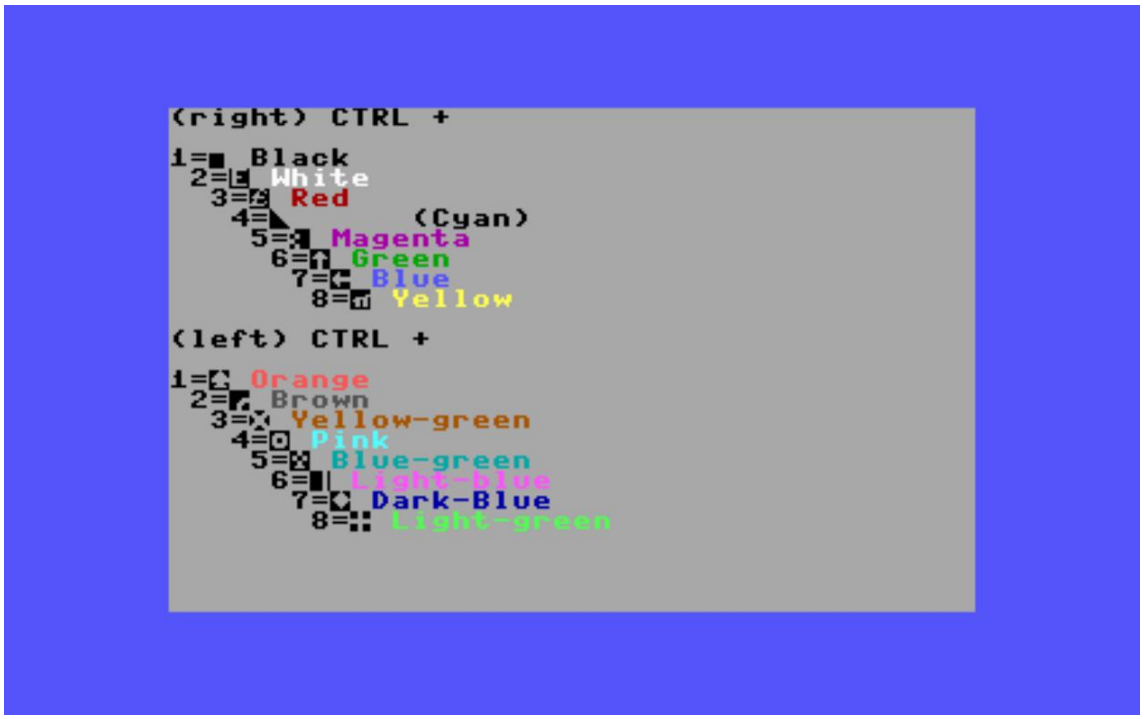

Figure 1.5: Color characters

eg.:
After printing "Blue" character on the screen, the text color is going to be blue.

1.6 Escape sequences

You can give some special commands to the typewriter, after pressing ESCAPE key on the keyboard:

Command	Effect
a	turn on Insert-mode
b	set the bottom-right corner of the textbox
c	turn off Insert-mode
i	insert a new line at the cursor
j	jump the first character of the line
k	jump to the last character of the line
l	turn on scrolling the screen ([0]:0548)
m	turn off scrolling the screen ([0]:0548)
n	set the normal textbox
o	turn off Insert-mode, invers and blink
p	delete characters to the cursor
q	delete characters from the cursor
r	decreasing the size of the screen
t	set the top-left corner of the textbox
v	scroll up the screen by one line
w	scroll down the screen by one line
x	quit from escape-sequence

Table 1.4: Escape-sequences

eg.:
Press 'v' after pressing ESCAPE –> the screen is scrolling up by one line

1.7 Special characters

You are able to make special functions with characters.

Figure 1.6: Special characters

eg.#1.:
After printing "[+]"(zoom in) character on the screen, the screen is zooming in.
eg.#2.:
After printing "[@]a"(escape+a) characters on the screen, INSERT-mode is on.
eg.#3.:
After printing "scnclr[r]"("scnclr"+enter) on the screen, the screen will be clean.

1.8 POKE & CHR characters

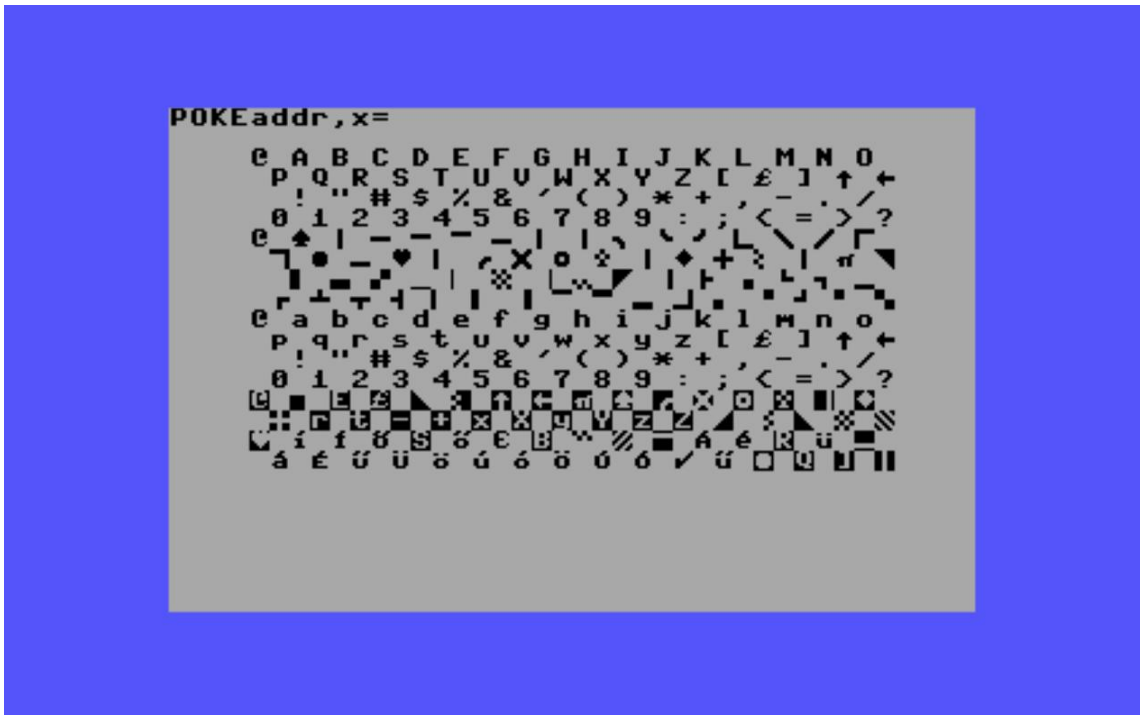

Figure 1.7: POKE characters

0	-> 228	8	-> 218	16	-> 252	24	-> 88
1	-> 211	9	-> 210	17	-> 253	25	-> 90
2	-> 212	10	-> 224	18	-> 254	26	-> 27
3	-> 213	11	-> 231	19	-> 255	27	-> 192
4	-> 214	12	-> 234	20	-> 65	28	-> 28
5	-> 215	13	-> 209	21	-> 81	29	-> 29
6	-> 216	14	-> 237	22	-> 83	30	-> 30
7	-> 217	15	-> 239	23	-> 87	31	-> 31

Table 1.5: CHR characters

...and CHR$(65)...CHR$(90) –> 1 ... 26!
...and CHR$(97)...CHR$(122) –> 129 ... 154!
eg.:
?CHR$(11) –> Blink on
?CHR$(16) –> Move cursor up by one
?CHR$(65) –> ”A”

1.9 Development environment

I am using right now the following configuration to develope. I have already tried TV4B on a lot of other configurations of course.

Name	Type
Monitor	22" LCD TV Samsung monitor
House	Corsair Carbide Series 200R Black
Power	Chieftec 650W CTG-650C 12cm Cable management box
Motherboard	ASUS M5A99X EVO R2.0
Processor	AMD FX-8350 AM3+ 4,0GHz (8cores)
Cooler	Antec Kühler H2O 620 Liquid Cooling System
Memory	4x4GB DDR3 2133MHz (Kingston HyperX XMP Blue)
Video Card	GIGABYTE GTX760 4GB DDR5 (GV-N760OC-4GD)
SSD	Samsung 250GB 2,5" SATA3 840 (MZ-7TE250BW)
HDD	WD 2TB 7200rpm SATA-600 64MB (Green WD20EZRX)
Optical Drive	LG-BH08 Blu-Ray (Super Multi) Pioneer BDR-208DBK DVD/BluRay writer
GamePad	Logitech RumblePad Dual-Action with ForceFeedback
Sound Card	Creative SB X-FI Titanium HD PCI-E (SB1270)
Speakers	Logitech Z906 THX quality sound system
Keyboard	A4-Tech KD-800L Backlight (blue)
Mouse	A4-Tech X-748K Anti-Vibrate USB
Operating Systems	Windows (XP,Vista,7,8,8.1: x86/x64) Linux (Debian, Ubuntu: x86/x64) (Wine)

Table 1.6: Development environment

1.10 TV4B

Turbo Vision 4 BASIC has got a parameter. This parameter is a name of a program or an encoded TVD. For example, let the program name is ”tetrisz”, so you can load it:

TV4B tetrisz.tv4

Another example, let the encoded TVD name is ”music”, so you can load it:

- TV4B music
- TV4B music.tvd

The two methodes above are equivalent with each other, if the ”music” and ”music.tvd” files are existed.

Chapter 2

BASIC

Well, this BASIC interpreter enables you to write programs for all purpose. You can write a program in BASIC or Assembly or both.

First of all, we have a closer look at the BASIC part. The memory is divided into 256 segments. The size of a segment is 64KiB. When you are writing a BASIC program, the code is getting into at the first segment of the memory.

The BASIC area[1] is located at [1]:0000 - [200]:FFFF. It consists of 200 segments, so you can use 200*65536byte = 12800KiB at least.

The rows of your program are to be numbered[2]. After typing a number as a line number, you can give a BASIC instruction. It is a program. The program is stored in memory, so you are able to list it with LIST command and to run it with RUN command, etc[3].

10 PRINT"Hello World!"

[1]You can modify the start of the BASIC area at [0]:002A, [0]:002B and [0]:002C.
[2]It can be 0...16777215. Current line number from [0]:0037 to [0]:0039.
[3]to save it with SAVE command, to load it with LOAD command

2.1 Variables

A variable is a part of the memory which contains a value of a number or a text. This value is able to be altered while running a program. Each variable is to be named. Give a ”speaking” name! For example: number or name.

Variables are located at [201]:0000 - [250]:FFFF. This area consists of 50 segments, so 50*65536 = 3200KiB. There are 6 different type of variables:

- Real for floating-point numbers (number=3.1415)
- String for text (name$=”Nagy Károly”)
- Integer for integer numbers (number%=-131072)
- Byte for byte (number!=64)
- Word for word (number@=32768)
- DWord for dword (number#=131072)

You can write the value of a variable on the screen with PRINT(?):
PRINTnumber
?name$

2.1.1 Real

Real variable stores a floating-point number in a range of -33554431...33554431.

Characteristic	**Sign**	**Mantissa**
6 bit	1 bit	25 bit

Table 2.1: Floating-point number (real variable)

eg.:
pi=3.1415 (pi=3.141499)
fpn=-6.283 (fpn=-6.28299)
a=15/2 (a=7)
a=15.0/2.0 (a=7.5)

2.1.2 String ($)

String variable stores a text. The max length of the text is 255 characters long.
eg.:
first$=”Nagy”
last$=”Károly”
name$=first$+” ”+last$

2.1.3 Integer (%)

Integer variable stores a 32 bit signed value in a range of -2147483647...2147483647.
eg.: num%=-33554432
num%=67108864

2.1.4 Byte (!)

Byte variable stores a 8 bit unsigned value in a range of 0...255.
eg.: num!=0 (num!=256)
num!=128

2.1.5 Word (@)

Word variable stores a 16 bit unsigned value in a range of 0...65535.
eg.: num@=0 (num@=65536)
num@=32768

2.1.6 DWord (#)

DWord variable stores a 32 bit unsigned value in a range of 0...4294967295.
eg.: num#=0 (num#=4294967296)
num#=2147483647

2.1.7 System variables

These variables are always alive and overwritable.

- pi (=3.1415926)
- ti (number of 1/100 seconds)
- ti$ (hhmmss, eg.: ”235935”)

2.1.8 How to name a variable?

You can use characters of 'a' to 'z', 'A' to 'Z' and '0' to '9' in the name of a variable. The first character of the name must not be a number!

2.2 Commands and instructions

Commands enable you to give a direct command to the interpreter (Type it in the typewriter and press ENTER). For example: LOAD, LIST, RUN, SAVE. Instructions enable you to write a BASIC program to instruct the interpreter what to do. For example: FOR, NEXT, GOSUB, RETURN. Some commands can be used in a program, too.

Most of all commands and instructions[4] have got some parameters which modify the original function.

2.2.1 AUTO

AUTO is a useful command for numbering program lines with an autoincrement value[5]. After executing "AUTO 10", each ENTER in a program makes the next line number with the previous line number add 10.

2.2.2 BOX

To draw a rectangle:
BOX<color>,<x1>,<y1>,<x2>,<y2>[,<d>,<p>]

color	0 - background, 1 - foreground, 4 - frame
x1	X coordinate of the left-top corner
y1	Y coordinate of the left-top corner
x2	X coordinate of the right-bottom corner
y2	Y coordinate of the right-bottom corner
d	degree of the rotation angle (0..359)
p	0: no paint; 1: paint

eg.:
BOX1,25,25,125,75 : draw a rectangle
BOX1,134,54,184,104,45,1 : draw a diamond

[4] the code of the previous and the last is at [0]:0096 and [0]:0097
[5] see the memory map at [0]:0073 and [0]:0074.

2.2.3 CHAR

To write a string to the screen:
CHAR<color>,<x>,<y>,<s>[,<i>]

color	0 - background, 1 - foreground, 4 - frame
x	X coordinate of the string (0..39)
y	Y coordinate of the string (0..24)
s	the string
i	inverse (0: normal; 1: inverse)

eg.:
CHAR1,23,12,"Nagy Károly"

2.2.4 CHDIR

To change directory:
CHDIR<"dirname">

dirname	the name of the directory to change

eg.:
CHDIR"programs": to change current directory to "programs"
CHDIR"..": to change up directory to the parent

2.2.5 CIRCLE

To draw a circle or a polygon[6]:

CIRCLE<color>,<x>,<y>,<r/w>[,<h>,<sa>,<ea>,<ra>,<pa>]

color	0 - background, 1 - foreground, 4 - frame
x	X coordinate of the shape
y	Y coordinate of the shape
w	radius or the width of the shape
h	the height of the shape
sa	start angle (0..359)
ea	end angle (0..359)
ra	rotate angle (0..359)
pa	polygon angle (0..359)

eg.:

CIRCLE1,159,99,50 : to draw a circle in the middle of the screen

CIRCLE1,159,99,50,25 : to draw an ellipse

CIRCLE1,159,99,50,,90,270 : to draw a half of a circle

CIRCLE1,159,99,50,,,,45,90 : to draw a rectangle rotating with 45 degree

CIRCLE1,159,99,50,,,,,120 : to draw a triangle

CIRCLE1,159,99,100,50,,,45 : to draw a rotated (by 45 degree) ellipse

2.2.6 CLOSE

To close an opened file[7]:

CLOSE<idnum>,<mode>

idnum	file handle number (0...255)
mode	0 - read, 1 - write

eg.:

CLOSE1,0 : to close the number 1 file (opened for reading)

CLOSE15,1 : to close the number 15 file (opened for writing)

2.2.7 CLR

You can clear all variables (names and values).

[6] see from [0]:02CC to [0]:02DB

[7] see the memory from [0]:025D.

2.2.8 CMD

To print what you see:

CMD<mode>

mode	**what's happend?**
0	print on screen
1	print on screen and to the printer
2	print on screen and write to file (CMDTV4B.TXT)

eg.: after executing CMD1...

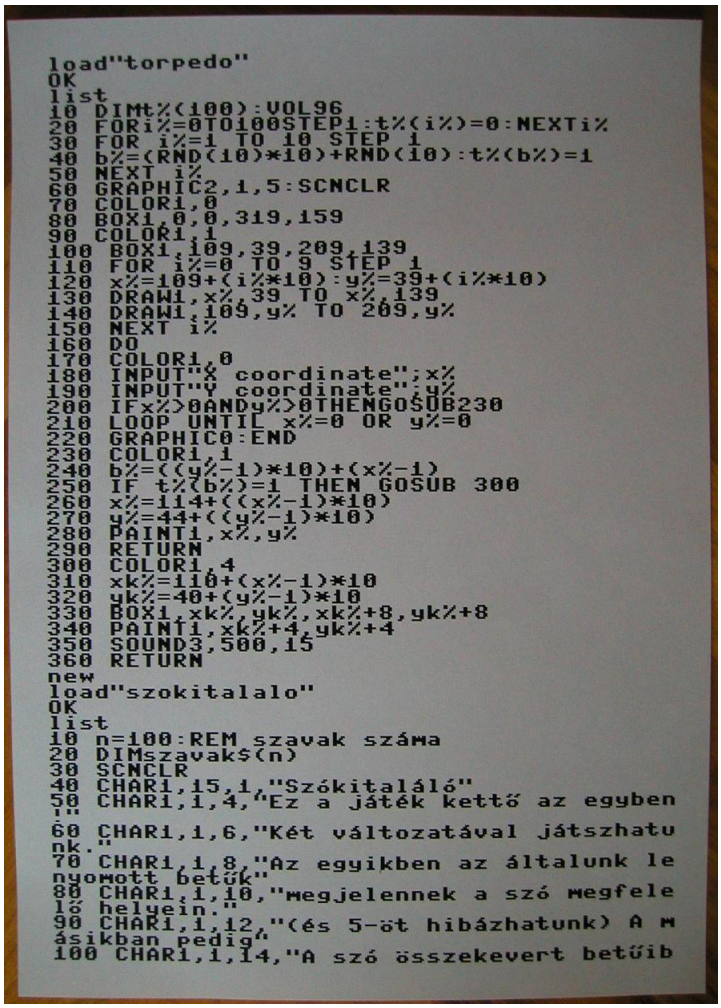

Figure 2.1: Print a LIST

If a printer page is full (58 rows), the current page is printing and a new page is going to be created. It also happens when you execute a CMD0 after a CMD1.

You can print a graphical screen, too. The first, the third and the fourth pictures were printed by a color Xerox Pharser 6280N PCL6 printer. The second was printed by a black&white HP LaserJet 3020 PCL5 printer.

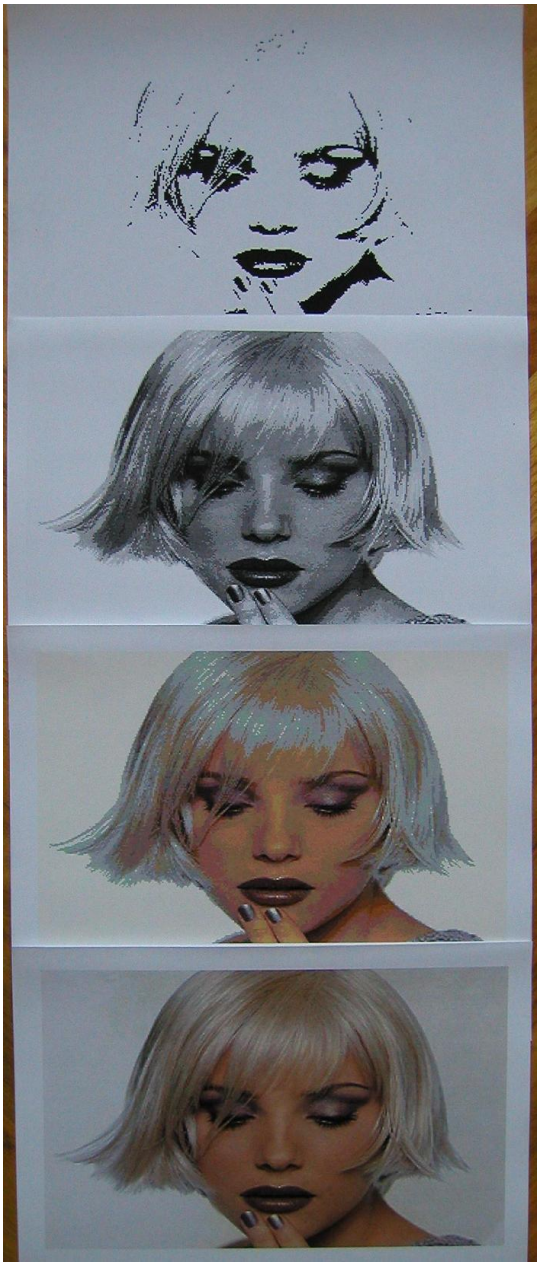

Figure 2.2: Print a picture (in 2, 256 and 16777216 colors)

2.2.9 COLOR

To set the color:
COLOR<color1>,<color2>

color1	0 - background, 1 - foreground, 4 - frame
color2	color index (0..255)

eg.:
COLOR0,4 : to set the background color to red
COLOR1,14 : to set the foreground color to yellow
COLOR4,15 : to set the frame color to white

2.2.10 COPY

To copy a file:
COPY<"filename1"> TO <"filename2">

filename1	the name of the file to copy
filename2	the name of the new file

eg.:
COPY"torpedo.tv4" TO "programs\torpedo.tv4" : to copy "torpedo"

2.2.11 DATA

To store datas[8]:
DATA<constant number and text separating with comma>

number	eg.: 800,10,550,25,...
text	eg.: "Nagy","Károly","CHARLEMAGNE",...

eg.:

```
10 DATA"Nagy Károly",32
20 READname$,age!
```

[8]see from [0]:003F to [0]:0043.

2.2.12 DEF FN

To make a macro:
DEFFN<name>=<macro>

name eg.: my or my(x)
macro eg.: RND(90) or RND(x)

eg.:

```
10 DEFFNmy(x)=RND(x)
20 PRINTFNmy(90)
```

2.2.13 DELETE

You are able to delete program lines.
DELETE<num>[-<num2>]

num program line number

eg.:
DELETE100 : delete the 100th line of the program
DELETE100- : delete the program lines after the 100th line
DELETE-100 : delete the program lines before the 100th line
DELETE100-200 : delete the program lines between 100 and 200

2.2.14 DIM

To create an array of variables:
DIM<name>(<size>[,<size2>,<size3>,<size4>])

name the name of a variable
size how many elements are in the array?
size2 2 dimension of array (up to 4)

eg.:
DIMreal(10) : to create a 10-elements of floating-point number array
DIMstr$(5) : to create a 5-elements of string array
DIMi%(15) : to create a 15-elements of integer array
DIMb!(60),w@(40),dw#(20) : to create a byte, word and dword array
DIMb!(4,3) : to create a 2 dimension array of bytes

2.2.15 DIRECTORY

It helps you to list the stored programs on the disk.
DIRECTORY[<str>,<num>]

str	a string with jokers eg.: "t?r*.tv4"
num	page number

eg.:
DIRECTORY : list all "Turbo Vision 4"(tv4) files
DIRECTORY"?o*.tv4" : eg. torpedo.tv4
DIRECTORY"*.*",2 : all files (the 2nd page)

2.2.16 DLOAD

It is just like LOAD.
DLOAD<"filename">

filename	the name of the file to load

eg.:
DLOAD"torpedo" : load "torpedo.tv4" to the memory

2.2.17 DO-LOOP-UNTIL/WHILE

To make an iteration:
DO <instr> LOOP UNTIL/WHILE <bool>

instr	instructions
bool	logical expression
UNTIL	iteration until bool is FALSE
WHILE	iteration while bool is TRUE

eg.:

```
10 i%=1
20 DO               : REM 20 DO:WHILE i%<11 ...and... 50 LOOP
30 PRINTi%          : REM 20 DO:UNTIL i%>10 ...and... 50 LOOP
40 i%=i%+1
50 LOOP UNTIL i%>10 : REM 50 LOOP WHILE i%<11
```

2.2.18 DRAW

To draw a pixel/line/polygon:
DRAW<color>,<x1>,<y1> [TO <x2>,<y2> [TO ...]]

color	0 - background, 1 - foreground, 4 - frame
x	X coordinate of a tip
y	Y coordinate of a tip

eg.:
DRAW1,50,50 : to draw a pixel
DRAW1,50,50 TO 100,100 : to draw a line
DRAW1,50,50 TO 100,50 TO 100,100 TO 50,100 TO 50,50 : to draw a rectangle
DRAW1,+59,-8 to 159,99: first set the cursor from the current position
DRAW1 TO 50;90 : to draw a 50-pixel-long-line from the previous position to the direction rotated with 90 degree

2.2.19 DSAVE

It is just like SAVE.
DSAVE<"filename">[,<num>]

filename	the name of the file to save
num	program line number (automatic start mode)

eg.:
DSAVE"program" : save the program from the memory to "program.tv4"
DSAVE"program",20 : save as automatic start with (D)LOAD (from 20th line)

2.2.20 END

The execution of a program is aborted by it.

2.2.21 EXIT

To exit from a FOR-NEXT or a DO-LOOP iteration:
EXIT
eg.:

```
10 i=2.5
20 DO
30 PRINTi
40 i=i+0.25
50 IF i>4.0 THEN EXIT
60 LOOP WHILE i<8.0
70 PRINT"end WHILE"
80 END
```

2.2.22 FOR-TO-STEP-NEXT

To make an iteration[9]:
FOR <var> TO <val> [STEP <val>] NEXT <var>

var iteration variable
val a value

eg.:

```
10 FOR i=15 TO 5 STEP -0.5
20 PRINTi
30 NEXTi
```

2.2.23 GET

To catch the pressed key[10]:
GET<var>

var variable name: number/string

eg.:

```
10 a$=""
20 GETa$:IFa$<>" "THENGOTO20
```

[9] see from [0]:0049 to [0]:004D
[10] ASCII code at [0]:00C6

2.2.24 GET#

To read a byte / char from a file:
GET#<idnum>,<var>

idnum file number handle (0...255)
var a number / string variable

eg.:

```
10 OPEN1,0,"datas.txt"
20 GET#1,a
30 PRINTa
40 CLOSE1,0
```

2.2.25 GETKEY

To wait a keypressing[11]:
GETKEY<var>

var variable name: number/string

eg.:

```
10 GETKEYa$
20 PRINTa$
```

2.2.26 GOSUB-RETURN

To call a subroutine:
GOSUB<num>

num a program line number where the subroutine starts

eg.:

```
10 SCNCLR
20 GOSUB50
30 PRINT"Two"
40 END
50 PRINT"One"
60 RETURN
```

[11] Virtual Key Code at [0]:07F6 and [0]:07F7

2.2.27 GOTO

To goto another line:
GOTO<num>

num a program line number where the program continues

eg.:

```
10 PRINT"Turbo Vision 4 BASIC"
20 GOTO10
```

2.2.28 GRAPHIC

To change the screen:
GRAPHIC<mode>[,<clear>,<share>]

mode 0-text; 1-bit/pixel; 2-byte/pixel; 3-4bytes/pixel
clear 0-no screen clear; 1-screen clear
share eg.: 5 - 5 text lines at bottom, see more detail at [0]:0083

eg.:

```
10 GRAPHIC1,1,5
20 CIRCLE1,159,99,25
30 PAINT1,159,99
```

2.2.29 IF-THEN-ELSE

To make a condition:
IF <bool> THEN <inst> [: ELSE <inst2>]

bool a logical expression
inst executed if bool is true
inst2 executed if bool is false

eg.:

```
10 INPUT"What is your age";age!
20 IF age!<19 THEN PRINT"CHILD" : ELSE PRINT"ADULT"
```

2.2.30 IMAGE

To view pictures[12]:

IMAGE<x>,<y>,<"filename">

x	X coordinate of the picture
y	Y coordinate of the picture
filename	the name of the picture

eg.:

```
10 GRAPHIC3
20 IMAGE0,0,"picture.jpg"
30 GETKEYa$
40 GRAPHIC0
```

2.2.31 INPUT

To get in datas:

INPUT["string";<var>]<var>[,<var2>,...]

string	an optional parameter
var	number or string variable

eg.:

```
10 INPUT"What is your name";name$
20 INPUT"What is your age";age!
30 PRINT"Give two numbers to add!"
40 INPUTa,b
50 PRINT"Sum is ";a+b
```

2.2.32 INPUT#

To get in datas[13]:

INPUT#<idnum>,<var>[,<var2>,...]

idnum	file handle number (0...255)
var	number or string variable

eg.:

```
10 OPEN1,0,"datas.txt"
20 INPUT#1,name$,age!
30 CLOSE1,0
```

[12] BMP (1,4,8,24bpp), PCX (1,4,8,24bpp) and (normal)JPEG

[13] see at [0]:0001.

2.2.33 HELP

To return with a short manual for instructions:
HELP[<"pattern">]

pattern the pattern of an instruction

eg.:
HELP : return with how to use HELP
HELP"re" : return with instructions have got a part of "RE"

2.2.34 KEY

You can define a mini programs for function keys. KEY1 for F1, KEY2 for F2, etc.
KEY[<idnum>,<miniprg>]

idnum the number of the function key (1...8)
miniprg a mini program to run (max 31 characters)

The basic settings:

- KEY1,SYS: REM 3-PLUS-1
- KEY2,DLOAD"
- KEY3,DIRECTORY
- KEY4,SCNCLR
- KEY5,DSAVE"
- KEY6,RUN
- KEY7,LIST
- KEY8,HELP

A mini program can be maximum 31 character long. If the last character of a mini program is " (eg.: DLOAD"), not executing automatically.
eg.:
KEY : list the stored mini programs[14]
KEY4,RUN20 : if you press F4 function key, the program runs from 20

[14][0]:055D = KEY1 , [0]:057D = KEY2 , [0]:059D = KEY3 , ... , [0]:063D = KEY8

2.2.35 LET

To give a value to a variable:
LET<var>=<expr>[:<var>=<expr>[:...]]

var number or string variable
expr an expression or a value

eg.:

```
10 LET name$="Nagy Károly":age!=32
```

2.2.36 LIST

You are able to list program lines.
LIST<num>[-<num2>]

num program line number

eg.:
LIST100 : list the 100th line of the program
LIST100- : list the program lines after the 100th line
LIST-100 : list the program lines before the 100th line
LIST100-200 : list the program lines between 100 and 200

2.2.37 LOAD

LOAD command loads a BASIC or an Assembly program from disk to the memory. It runs the program, if you save beforehand it with automatic start mode.
LOAD<"filename">

filename the name of the file to load

eg.:
LOAD"torpedo" : load "torpedo.tv4" to the memory

2.2.38 LOCATE

To locate the graphical cursor:
LOCATE<x>,<y>
LOCATE<d>;<a>

- **x** X coordinate of graphical cursor
- **y** Y coordinate of graphical cursor
- **d** how far from the current position
- **a** rotation angle

eg.:

```
10 GRAPHIC2,1,5
20 DRAW1,50,50
30 LOCATE100,100
40 TO 200,50
```

eg.:
LOCATE50;45: move the cursor 50 pixels far rotating with 45 degree
LOCATE-59,+1: move the cursor from the current position

2.2.39 MKDIR

To make a directory:
MKDIR<"dirname">

- **dirname** the name of the directory to make

eg.:
MKDIR"programs": to make "programs" directory

2.2.40 MONITOR

You can write assembly programs in the MONITOR.
eg.:
MONITOR : entering another editor[15] where you can write assembly programs

[15] see at [0]:054B.

2.2.41 MUSIC

You are able to play[16] a music.
MUSIC[<"filename">,<num>]

filename	the name of the file (eg.: music.mod)
num	0-normal; 1-loop (WAV and MP3)

eg.:
MUSIC"music.wav" : play "music.wav"
MUSIC"music.mod" : play "music.mod" (in loop)
MUSIC"music.mp3",1 : play "music.mp3" (loop)
MUSIC : play "silence" (stop the music)

2.2.42 NEW

It enables you to clear the whole program and all variables from the memory.

2.2.43 ON

To make a choice:
ON <var/const> GOTO<num>,<num2>,...
ON <var/const> GOSUB<num>,<num2>,...

var	a number variable
const	a constant number
num	program line number

eg.:

```
10 INPUT"Choice";a
20 ON a GOTO40,50,60
30 PRINT"0":END
40 PRINT"1":END
50 PRINT"2":END
60 PRINT"3":END
```

[16] at [0]:0080 0:nothing; 1:WAV; 2:MOD; 3:MP3

2.2.44 OPEN

To open a file[17]:
OPEN<idnum>,<mode>,<filename>

idnum	file handle number (0...255)
mode	0-open for read; 1-open for write
filename	the name of the file to open

eg.:

```
10 OPEN1,1,"datas.txt"
20 CLOSE1,1
```

2.2.45 PAINT

To paint a closed polygon:
PAINT<color>,<x>,<y>

color	0 - background, 1 - foreground, 4 - frame
x	X coordinate
y	Y coordinate

eg.:

```
10 GRAPHIC3,1,5
20 BOX1,149,89,169,109
30 PAINT1,159,99
```

2.2.46 POKE

To put a byte to the memory:
POKE<addr>,<value>

addr	address of memory (0...65535)
value	value (0..255)

eg.:
POKE3072,1 : to put an 'A' letter at the top-left corner of the screen

[17] see the memory from [0]:025D.

2.2.47 PRINT

To write something on the screen[18]:
PRINT<const>/<var>/<expr>

const	constant of a number or string
var	variables
expr	expressions

eg.:
PRINT"Hello World!" : to write 'Hello World!' at the cursor position
PRINTage! : to write age! byte-type variable on the screen
PRINTa%+b% : to write the sum of a% and b% integer-type variables
PRINTa@,b@ : to write a and b word-type variables separatelly
PRINTa#;b# : to write a# and b# dword-type variables unseparatelly
PRINTPEEK(3072) : to write the character at the top-left corner of the screen

2.2.48 PRINT#

To put something to a file[19]:
PRINT#<idnum>,<var>[,<var2>,...]

idnum	file handle number (0...255)
var	number or string variables

eg.:

```
10 OPEN1,1,"datas.txt"
20 LET name$="Nagy Károly":age!=32
30 PRINT#1,name$,age!
40 CLOSE1,1
```

2.2.49 READ

To read datas to variables:
READ<var>[,<var2>,...]

var	number or string variables

eg.:

```
10 READname$,age!
20 DATA"Nagy Károly",32
```

[18][0]:00CA - X coordinate and [0]:00CD Y coordinate of the cursor
[19]see at [0]:0001.

2.2.50 REM

To make a remark:
REM<anything>

anything constant/number/string/variable/...

eg.:

```
10 REM ************************
20 REM * Turbo Vision 4 BASIC *
30 REM *           By         *
40 REM *      Károly Nagy     *
50 REM *     (CHARLEMAGNE)    *
60 REM ************************
70 REM :
80 SCNCLR : REM SCreeN CLeaR
...
```

2.2.51 RENAME

To rename a file:
RENAME<"oldfilename"> TO <"newfilename">

oldfilename the name of the file to rename
newfilename the name of the new file

eg.:
RENAME"torpedo.tv4" TO "game.tv4" : to rename "torpedo"

2.2.52 RENUMBER

To renumber[20] a BASIC program:
RENUMBER[<num1>[,<num2>]]

num1 the first line number (0...16777215)
num2 the increment value (0..16777215)

eg.:
RENUMBER 50,20 : to renumber the BASIC lines –> 50, 70, 90, 110, ...

[20] see at [0]:0003

2.2.53 RESTORE

To research DATA lines for READ:
RESTORE[<num>]

num a program line number DATA from

eg.:

```
10 READname$,age!
20 RESTORE:END
30 DATA"Nagy Károly",32
```

2.2.54 RUN

The stored program is executed.
RUN[<num>]

num a program line number run from

eg.:
RUN : run the program from the first line
RUN50 : run the program from the 50th line

2.2.55 SAVE

SAVE command saves the BASIC program to the disk.
SAVE<"filename">[,<num>]

filename the name of the file to save
num program line number (automatic start mode)

eg.:
SAVE"program" : save the program from the memory to "program.tv4"
SAVE"program",20 : save as automatic start with (D)LOAD (from 20th line)

2.2.56 SCNCLR

You can clear the screen.
eg.:
SCNCLR : clear the screen[21]

[21] to fill up the memory from [0]:0C00 to [0]:0FE7 with 32(SPACE)

2.2.57 SCRATCH

To remove a file[22] from the current directory:
SCRATCH<"filename">

filename the name of the file to delete

eg.:
SCRATCH"datas.txt" : to delete the "datas.txt" file

2.2.58 SCREEN

To translate and rotate the screen in 3D:
SCREEN<tz>,<rx>,<ry>,<rz>

tz Translate Z (0...535)
rx Rotate X (0...359)
ry Rotate Y (0...359)
rz Rotate Z (0...359)

eg.:
SCREEN200,0,45,0 : to rotate the screen about Y axis (45 degree)

2.2.59 SEGMENT

To set the DATA segment (from / to where you read / write datas):
SEGMENT<num>

num the segment number (0...255)

eg.:
SEGMENT0 : to set DS to 0

[22] or an empty directory

2.2.60 SOCKET

To make a TCP or an UDP ethernet connections[23]:
SOCKET<"message">[,<"ipaddr">]

message	the message to send (eg.: "Hello World!")
ipaddr	IP address to send (eg.: "192.168.2.102")

eg.:
SOCKET"Hello World!" : send "Hello World!" to itself

2.2.61 SOUND

To make a sound[24]: **SOUND<channel>,<Hz>,<delay>**

channel	1-1st; 2-2nd; 3-3rd channel
Hz	Value of sound (0...1023)
delay	delay to hear the sound (60 is a second)

Note	V	Hz	V	Hz	V	Hz	V	Hz
A	7	110	516	220.2	770	440.4	897	880.7
B	118	123.5	571	246.9	798	494.9	911	989.9
C	169	130.8	596	261.4	810	522.7	917	1045
D	262	146.8	643	293.6	834	588.7	929	1177
E	345	164.7	685	330	854	658	939	1316
F	383	174.5	704	349.6	864	699	944	1398
G	453	195.9	739	392.5	881	782.2	953	1575

Table 2.2: Sound notes

eg.:
SOUND1,897,30 : to make a sound on the first channel (880.7Hz and 1/2second)
SOUND2,571,60 : to make a sound on the second channel (246.9Hz and 1second)
SOUND3,917,15 : to make a sound in the third channel (1045Hz and 1/4second)

[23] see memory from [0]:065D to [0]:0769
[24] see memory at [0]:0078 for Mono/Stereo

2.2.62 SSHAPE-GSHAPE

To store a sprite:

SSHAPE<var>,<x1>,<y1>,<x2>,<y2>

var	a name of a variable to identify the sprite
x1	X coordinate of the left-top corner
y1	Y coordinate of the left-top corner
x2	X coordinate of the right-bottom corner
y2	Y coordinate of the right-bottom corner

To put a sprite:

GSHAPE<var>,<x>,<y>[,<mode>]

var	a name of a variable to identify the sprite
x1	X coordinate of the left-top corner
y1	Y coordinate of the left-top corner
mode	0-normal; 1-invers(only bit/pixel); 2-OR; 3-AND; 4-XOR;

eg.:

```
10 GRAPHIC2,1,5
20 BOX1,149,89,169,109
30 COLOR1,4
40 CIRCLE1,159,99,9
50 PAINT1,159,99
60 SSHAPEspr$,149,89,169,109
70 GSHAPEspr$,10,10
```

2.2.63 STOP-CONT

When you stop a running program with STOP instruction, you have an opportunity to continue with CONT command.

2.2.64 SYS

To run an assembly program[25]:
SYS<num>

num a decimal number (0...65535) where an assembly program starts

eg.:
SYS52650 : to run a secret code[26] in the memory

2.2.65 TRAP-RESUME

To make a trap[27] for errors:
TRAP<num/var>

num a BASIC line number to go to if an error occured

eg.: (when you press ESCAPE, goto 50)

```
10 TRAP50
20 PRINT"Hello"
30 GOTO20
40 END
50 PRINT"This is a TRAP!"
60 INPUT"Do you want to abort (y/n)";a$
70 IF a$="y" THEN END
80 RESUME
```

2.2.66 TRON-TROFF

You can turn on / off debug. It shows the program line numbers while running a BASIC program.

[25]the registers (AR, XR, YR, SP) are from [0]:07F2 to [0]:07F5
[26]at [0]:CDAA is the details of the author
[27]more details from [0]:04EF to [0]:04F7

2.2.67 USING-PUDEF

To format the writing[28]:
USING<format>

format a pattern to form, eg.: "$##.#"

To rewrite the symbols[29] used by USING:
PUDEF<" ,.$">

[SPACE] at the first place
[,] at the second place
[.] at the third place
[$] at the fourth place

eg.:

```
 10 OPEN1,1,"using.txt"
 20 x=5.25
 30 PUDEF" ,."
 40 PRINT#1 USING"$###.##";x,43.768,123
(40 PRINT USING"$###.##";x,43.768,123)
 50 CLOSE1,1
```

2.2.68 VERIFY

You can set the program in the memory against a program on the disk.
VERIFY<"filename">

filename the name of the file to verify

eg.:
VERIFY"torpedo" : compare the program in the memory with torpedo(.tv4)

[28] You can use with "PRINT" for the screen and "PRINT#" for file
[29] see memory map from [0]:04E7 to [0]:04EA

2.2.69 VOL

VOL stands for volume[30]. It turns on or off the sound channel. To set the volume:
VOL<num/var>

num a number (0...8 or 9...255)

eg.:
VOL0 : mute the sounds
VOL255 : maximum volume (or VOL8)

2.2.70 WAIT

To wait for a logical expression.
WAIT<addr/var>,<val1/var>[,<val2/var>]

addr an address (0...65535) to a value(x)
val1 a value (0..255) AND x
val2 a value (0..255) XOR x

eg.:
WAIT198,32 : to wait for keypressing

[30] see the memory map at [0]:007E

2.3 Functions

Functions are so procedures which return a value according to the given parameters. They are called recursively of course.

2.3.1 ABS

To return with absolute value:
ABS(<num/var>)

num if num>0 then return num else -1*num

eg.:
ABS(5) : return with 5
ABS(-2.76) : return with 2.7599

2.3.2 ASC

To return with ASCII code[31]:
ASC(<str/var>)

str a string

eg.:
ASC(”A”) : return with 1
ASC(”Ab”) : return with 1

[31]You can set the current characters-set at [0]:07FF (the 1st bit)

2.3.3 ATN

To return with arcus tangent:
ATN(<num/var>)

num a degree (0..359)

eg.:
ATN(0) : return with 0
ATN(90) : return with 1.5596856

2.3.4 CHR$

To return with a character:
CHR$(<num/var>)

num a number

eg.:
CHR$(65) : return with 'A'
CHR$(97) : return with 'a'

2.3.5 COS

To return with cosinus:
COS(<num/var>)

num a degree (0..359)

eg.:
COS(0) : return with 1
COS(90) : return with -0.448

2.3.6 DEC

To return with decimal number:
DEC(<hexnumstr/var>)

hexnumstr a hexadecimal number given in a string

eg.:
DEC("A") : return with 10
DEC("CDAA") : return with 52650

2.3.7 ERR$

To return with the last error message:
ERR$(<num/var>)

num the number of the error

eg.:
ERR$(11) : return with "?SYNTAX ERROR" if [0]:00B1[32] is zero

2.3.8 EXP

To return with exponent of "e"(=2.71828183):
EXP(<num/var>)

num the power of "e"

eg.:
EXP(2) : return with 7.384 (e^2)

2.3.9 FN

To return with user defined function value:
FN<funcname>

funcname the macro name which you define with DEF instruction

eg.#1.: (return with a random number between 0 and 89)

```
10 DEFFNmy=RND(90)
20 PRINTFNmy
```

eg.#2.: (return with a random number between 0 and 44)

```
10 DEFFNmy(x)=RND(x)
20 PRINTFNmy(45)
```

[32]You can read error messages in the memory map.

2.3.10 FRE

It shows the free bytes of BASIC memory.
FRE(<num/var>)

num 0 - get the free memory in bytes

eg.:
FRE(0) : return with the free memory in bytes

2.3.11 HEX$

To return with hexadecimal number:
HEX$(<num/var>)

num a decimal number

eg.:
HEX$(10) : return with 'A'
HEX$(52650) : return with 'CDAA'

2.3.12 INSTR

To return with a substring position:
INSTR(<str/var>,<substr/var>)

str a string in which is looking for substr
substr a string which is searched in str

eg.:
INSTR("There is a cat in the hat","cat") : return with 12

2.3.13 INT

To return with an integer:
INT(<num/var>)

num a number

eg.:
INT(5) : return with 5
INT(-2.76) : return with -2

2.3.14 JOY

To return with the state of a joystick[33]:
JOY(<num/var>)

num 1-the 1st joystick, 2-the 2nd joystick, ...

8=North-West	**1**=North	**2**=North-East
7=West	+**128**=FIRE	**3**=East
6=South-West	**5**=South	**4**=South-East

Table 2.3: Joystick state

eg.:
JOY(1) : return with the state of the first joystick
In that case you have not got any joystick, you can emulate It...

NUM7=North-West	**NUM8**=North	**NUM9**=North-East
NUM4=West	**NUM5**=FIRE	**NUM6**=East
NUM1=South-West	**NUM2**=South	**NUM3**=South-East

Table 2.4: Emulate joystick with numeric keypad

2.3.15 LEFT$

To return with a substring on the left:
LEFT$(<str/var>,<num/var>)

str a string which is cut by num on the left
num how many characters are cut on the left

eg.:
LEFT$("CHARLEMAGNE",6) : return with 'CHARLE'

[33]or GamePad with Dual Action + Force Feedback: [0]:07FA,[0]:07FB,[0]:07FC

2.3.16 LEN

To return with the length of a string:
LEN(<str/var>)

str a string

eg.:
LEN("CHARLEMAGNE") : return with 11

2.3.17 LOG

To return with logaritm:
LOG(<num/var>)

num a number

eg.:
LOG(5) : return with 1.6
LOG(2) : return with 0.6931471

2.3.18 MID$

To return with a substring:
MID$(<str/var>,<num1/var>[,<num2/var>])[=<str/var>]

str a string
num1 where
num2 how many (optional)

eg.:
MID$("CHARLEMAGNE",5,4) : return with 'LEMA'
MID$("CHARLEMAGNE",5) : return with 'LEMAGNE'
if a$="CHARLEMAGNE" and MID$(a$,5)="ok" : a$="CHARokMAGNE"

2.3.19 PEEK

To return with a byte from memory:
PEEK(<num/var>)

num a number points to the memory (0...65535)

eg.:
PEEK(3072) : return with the first character of the screen ([0]:0C00)

2.3.20 POS

It shows the next print position.
eg.:
POS : return with the column where the next character is appeared

2.3.21 RCLR

To return with a color index:
RCLR(<num/var>)

num 0-BackGround; 1-ForeGround; 4-Frame

eg.:
RCLR(4) : return with the frame color index

2.3.22 RDOT

To return with X or Y coordinate or color index:
RDOT(<num/var>)

num 0-X position; 1-Y position; 2-color index

eg.:
RDOT(1) : return with the row number of the cursor (on text screen)
RDOT(1) : return with the row number of the cursor (on graphic screen)

2.3.23 RGR

To return with the current graphic mode or screen mode:
RGR(<num/var>) eg.:

num	return	remark
0	0	text screen
0	1	bit/pixel
0	2	byte/pixel (256color)
0	3	4bytes/pixel (TrueColor)
1	see the screen modes	at [0]:0083

Table 2.5: Graphic modes

RGR(0) : return with the current graphic mode
RGR(1) : return with the screen mode (see at [0]:0083)

2.3.24 RIGHT$

To return with a substring on the right:
RIGHT$(<str/var>,<num/var>)

str a string which is cut by num on the right
num how many characters are cut on the right

eg.:
RIGHT$("CHARLEMAGNE",5) : return with 'MAGNE'

2.3.25 RND

To return with a random number:
RND(<num/var>)

num the maximum random number + 1

eg.:
RND(90) : return with a random number between 0 and 89
RND(0) : return with a random number between 0 and 1

2.3.26 SGN

To return with sign:
SGN(<num/var>)

num a number

eg.:
SGN(5) : return with 1
SGN(0) : return with 0
SGN(-4) : return with -1

2.3.27 SIN

To return with sinus:
SIN(<num/var>)

num a degree (0..359)

eg.:
SIN(0) : return with 0
SIN(90) : return with 0.8939966

2.3.28 SOCKET

To return with the received message from ethernet[34]:
SOCKET
eg.:
SOCKET : to return the received buffer (see [0]:0660))

2.3.29 SPC

To return with spaces:
SPC(<num/var>)

num how many spaces?

eg.:
SPC(5) : return with 5 space

[34] see memory from [0]:065D to [0]:0769

2.3.30 SQR

To return with square root:
SQR(<num/var>)

num a number

eg.:
SQR(16) : return with 4
SQR(2) : return with 1.4142135

2.3.31 STR$

To return with a string:
STR$(<num/var>)

num a number

eg.:
STR$(56) : return with '56'

2.3.32 TAB

To return with setting cursor:
TAB(<num/var>)

num which columns?

eg.:
TAB(5) : setting the column of the cursor to the 5th column

2.3.33 TAN

To return with tangent:
TAN(<num/var>)

num a degree (0..359)

eg.:
TAN(0) : return with 0
TAN(90) : return with -1.9952

2.3.34 USR

To return with a value[35] created by user defined assembly program[36]:
USR(<num/var>)

num an integer value

eg.:

```
POKE1282,80  : REM the start address (seg=0; low=0; high=50;)
POKE20480,96 : REM the assembly program from 0x5000 (=RTS(0x60))
a%=USR(6)
```

2.3.35 VAL

To return with the value:
VAL(<str/var>)

str a number given in a string

eg.:
VAL("123.456") : return with 123.456
VAL("39 A 17") : return with 39
VAL("A 293") : return with 0

[35]at [0]:0061 (this is also the USR parameter)
[36]the address of the assembly program at [0]:0500 (seg), [0]:0501 (low), [0]:0502 (high)

Chapter 3

Assembly

Well, this BASIC interpreter enables you to write programs for all purpose. You can write a program in BASIC or Assembly or both.

Second of all, we have a closer look at the Assembly part. Assembly uses mnemonics to instruct the computer what to do. A mnemonic is a part of a meaningful English word[1].

The rows of your program are to be numbered[2]. After typing a number as a line number, you can give an Assembly mnemonic. It is a program. The program is stored in memory, so you are able to list it with D command and to run it with G command, etc[3].

```
A 5000 A9 08    LDA #$08
A 5002 8D 00 0C STA $0C00
A 5005 A9 85    LDA #$85
A 5007 8D 01 0C STA $0C01
A 500A A9 8C    LDA #$8C
A 500C 8D 02 0C STA $0C02
A 500F 8D 03 0C STA $0C03
A 5012 A9 8F    LDA #$8F
A 5014 8D 04 0C STA $0C04
A 5017
G5000
```

[1] eg.: LDA stands for LoaD Accumulator, BRK stands for BReaK, ...

[2] It can be 0000...FFFF.

[3] to save it with S command, to load it with L command

3.1 MONITOR

MONITOR is an embeded editor for dumping (view / alter) the memory, writing / modifying / listing Assembly programs, looking for a byte in memory, etc.

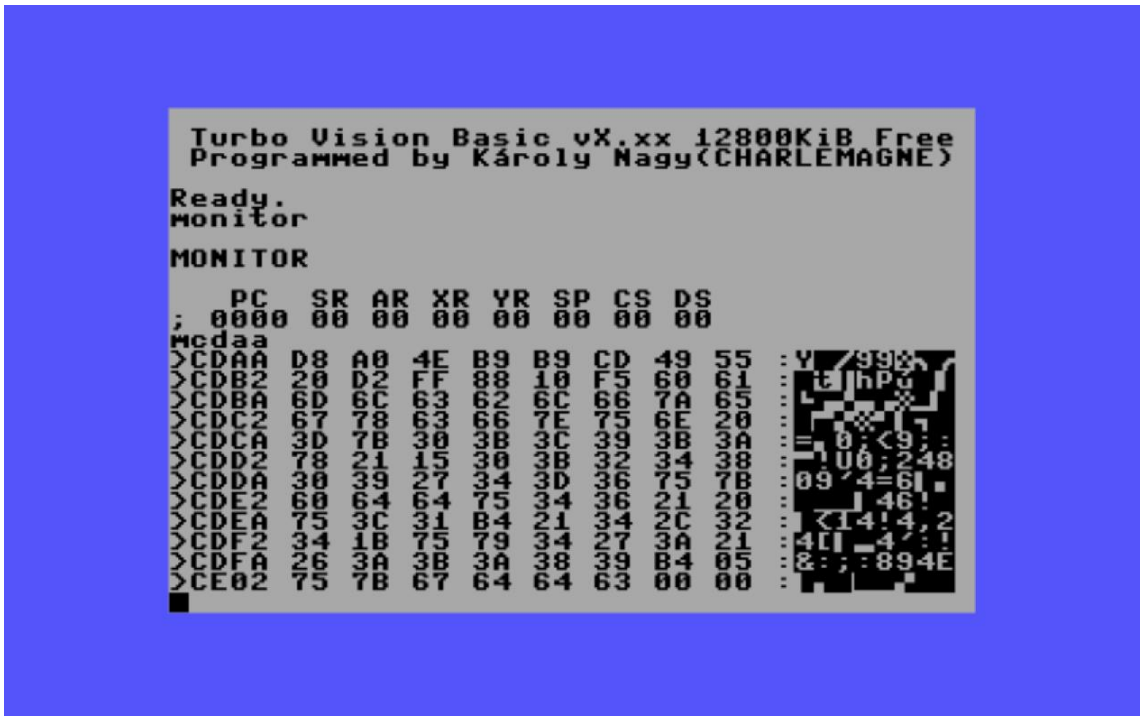

Figure 3.1: MONITOR: MCDAA

There are 8 registers[4]:

PC	Program Counter for where you are (16bit)
SR	Status Register for status (8bit)
AR	Accumulator Register for storing current data (8bit)
XR, YR	X Register, Y Register for help indexing (8bit)
SP	Stack Pointer for stack position (8bit)
CS	Code Segment for where the program is (8bit)
DS	Data Segment for where the reading / writing data is (8bit)

Table 3.1: Registers

[4]Only some registers are directly available: AR, XR and YR

3.2 Registers

A register is a memory. It is fast and easy-to-use. As you see the previous page TV4B uses 8 registers in MONITOR. Each register has got a function.

3.2.1 PC (=Program Counter)

PC shows you where the next instruction is executed. It is not directly available for using. PC is a 16 bit register so its value can be from 0 to 65535.

3.2.2 SR (=Status Register)

SR shows you the current status of the current executed expression. It is not directly available for using. SR is an 8 bit register so its value can be from 0 to 255.

3.2.3 AR (=Accumulator Register)

AR stores the current result of an aritmetic expression or other (eg.: status). It is directly available for manipulating. AR is an 8 bit register so its value can be from 0 to 255.
eg.: (the value of AR is 31)

```
A 5000 A9 1F    LDA #$1F
```

3.2.4 XR and YR (=X,Y Register)

These are general-purpose registers. They are directly available for manipulating. They are both 8 bit registers so their value can be from 0 to 255.
eg.: (the value of XR is 31 and YR is 63)

```
A 5000 A2 1F    LDX #$1F
A 5002 A0 3F    LDY #$3F
```

3.2.5 SP (=Stack Pointer)

SP shows you the current position of the stack[5]. It is not directly available for using. SP is an 8 bit register so its value can be from 0 to 255.
eg.: (using stack for storing) –> see at "Stack instructions..." section

3.2.6 CS (=Code Segment)

CS shows you where the program code is. It is not directly available for using. CS[6] is an 8 bit register so its value can be from 0 to 255.

3.2.7 DS (=Data Segment)

DS shows you where the program is getting or putting datas. It is not directly available for using. DS[7] is an 8 bit register so its value can be from 0 to 255.

[5]Stack is a LIFO (Last-In-First-Out), see the memory at [0]:0100 (System Stack)
[6]see the memory at [0]:07FE
[7]see the memory at [0]:07FD

3.3 Commands in MONITOR

The most of all commands in MONITOR are only one letter.

3.3.1 . (=Assembler)

It is just like A. You can write an assembly instruction into the memory.
eg.:

```
.5000 LDA #$01
```

3.3.2 > (=Alter bytes)

See the 3.1. Figure again! It enables you to alter bytes in the memory.

3.3.3 A (=Assembler)

It is just like .. You can write an assembly instruction into the memory.
eg.:

```
A5000 LDA #$01
```

3.3.4 C (=Compare)

You can compare datas.
C <hexnum1> <hexnum2> <hexnum3>

hexnum1 (from) the start address (0000...FFFF)
hexnum2 (from) the end address (0000...FFFF)
hexnum3 (to) the start address (0000..FFFF)

eg.:
C 5000 5029 6000 : to compare datas between 5000 and 5029 to 6000.

3.3.5 D (=Disassembler)

You can list an assembly program. It is called disassembler.

D <hexnum1> [<hexnum2>]

hexnum1 (from) the start address (0000...FFFF)

hexnum2 (to) the end address (0000...FFFF)

Figure 3.2: DISASSEMBLER: DCDAA

eg.:

D 5000 5011 : to disassemble datas between 5000 and 5011

3.3.6 F (=Fill)

You can fill the memory with a byte.

F <hexnum1> <hexnum2> <hexnum3>

hexnum1 the start address (0000...FFFF)
hexnum2 the end address (0000...FFFF)
hexnum3 the byte (00..FF)

eg.:
F 0C00 0C10 01 : to fill the text screen at the beginning with 17 'A'.

3.3.7 G (=Go)

You can run an assembly program.

G<hexnum>

hexnum a number in hexadecimal

eg.:
GCDAA : to run an assembly program from CDAA

3.3.8 H (=Hunt)

You can hunt a byte or bytes in memory.

H <hexnum1> <hexnum2> <hexnum3> [<hexnum4> ...]

hexnum1 (from) the start address (0000...FFFF)
hexnum2 (from) the end address (0000...FFFF)
hexnum3 the byte you want to hunt

eg.:
H 0C00 0CFF 02 81 : to hunt 'Ba' in the text screen.
H 0C00 0CFF "Basic" : to hunt 'Basic' in the text screen.

3.3.9 L (=Load)

You can load an assembly program.

L"filename"

filename the name of the file to load

eg.:
L"perfect" : to load the perfect.tv4 file into the memory

3.3.10 M (=Memory Map View)

You can view the memory. See the 3.1.Figure again!

M<hexnum>

hexnum	a number in hexadecimal

eg.:

MCDAA : to view the memory from CDAA

3.3.11 R (=Registers)

You can view the value of the registers:

R

eg.:

R : to view the registers and their current values

3.3.12 S (=Save / Store)

You can save an assembly program.

S"filename",<seg1>,<ofs1>,<seg2>,<ofs2>[,<seg3>,<ofs3>]

filename	the name of the file to save
seg1	start segment (0...255)
ofs1	start offset (0000...FFFF)
seg2	end segment (0...255)
ofs2	end offset (0000...FFFF)
seg3	autostart segment (0...255)
ofs3	autostart offset (0000...FFFF)

eg.:

S"perfect",0,5000,0,5023,0,5000 : to save the perfect.tv4 (with autostart mode)

3.3.13 T (=Transfer)

You can transfer datas.

T <hexnum1> <hexnum2> <hexnum3>

hexnum1	(from) the start address (0000...FFFF)
hexnum2	(from) the end address (0000...FFFF)
hexnum3	(to) the start address (0000..FFFF)

eg.:

T 5000 5029 6000 : to transfer datas from 5000-5029 to 6000.

3.3.14 V (=Verify)

You can verify an assembly program.

V"filename"

filename	the name of the file to verify

eg.:

V"interrupt" : to verify the interrupt.tv4 file with the program in memory

3.3.15 X (=eXit from MONITOR)

You quit to BASIC editor:

X

eg.:

X : to exit from MONITOR.

3.4 Addressing modes

Addressing mode is that how you want to access datas in memory.

3.4.1 Immediate / literal

It enables you to give a value to a register.
eg.:

```
A 5000 A9 01    LDA #$01
A 5002 A2 02    LDX #$02
A 5004 A0 03    LDY #$03
```

3.4.2 Absolute / Direct

It enables you to give a value (stored in address) to a register.
eg.:

```
A 5000 AD 00 0C LDA $0C00
A 5003 AE 01 0C LDX $0C01
A 5006 AC 02 0C LDY $0C02
```

3.4.3 Absolute / Direct-indexed

It enables you to give a value (stored in address + index) to a register.
eg.:

```
A 5000 A2 01    LDX #$01
A 5002 A0 02    LDY #$02
A 5004 BD 00 0C LDA $0C00,X
A 5007 B9 00 0C LDA $0C00,Y
A 500A BE 00 0C LDX $0C00,Y
A 500D BC 00 0C LDY $0C00,X
```

3.4.4 Relative / Indirect

It enables you to give a value (stored in address + X (is another address)) to a register.
eg.:

```
A 5000 A2 00    LDX #$00
A 5002 A1 D0    LDA ($D0,X)
```

3.4.5 Relative / Indirect-indexed

It enables you to give a value (stored in address (is another address) + Y) to a register.
eg.:

```
A 5000 A0 2A    LDY #$2A
A 5002 B1 D0    LDA ($D0),Y
```

3.5 Flags

A flag shows what happend after you executed an Assembly instruction.

bit	7	6	5	4	3	2	1	0
name	N	V		B	D	I	Z	C

Table 3.2: Flags

3.5.1 N (=Negative)

If the 7th bit of the result of the last operation is 1, this flag is 1.

3.5.2 V (=oVerflow)

If an overflow happens from the 6th bit to the 7th bit of the result of the last operation, this flag is 1.

3.5.3 B (=Break)

If a program makes an interrupt, this flag is 1.

3.5.4 D (=Decimal)

If you want to make aritmetic calculations in decimal number system, this flag is 1.

3.5.5 I (=Interrupt)

If an interrupt exists, this flag is 1.

3.5.6 Z (=Zero)

If the result of the last operation is 0, this flag is 1.

3.5.7 C (=Carry)

If the result of the last operation cannot be stored on 8 bits, this flag is 1.

3.6 Assembly instructions

Each Assembly instruction is a part of a meaningful English word to instruct the computer what to do. There are instructions for data loading (LDA, LDX, LDY), data storing (STA, STX, STY), data transfering (TAX, TAY, TXA, TYA), etc.

3.6.1 Arithmetic instructions: ADC, SBC

You can add numbers with ADC[8], and subtract from each other with SBC[9].
Flags: NVZC
eg.: (writing 'Q'(17th) on the top-left of the screen)

```
A 5000 A9 0F    LDA #$0F
A 5002 69 02    ADC #$02
A 5004 8D 00 0C STA $0C00
```

3.6.2 Bit manipulating instructions: ASL, LSR, ROL, ROR

You can shift bits on the left by 1 with ASL[10], and on the right by 1 with LSR[11]. You can rotate bits on the left by 1 bit with ROL[12] and on the right by 1 bit with ROR[13].
Flags: NZC
eg.: (64 multiply by 2)

```
A 5000 A9 40    LDA #$40
A 5002 0A       ASL
```

[8] ADd with Carry to accumulator
[9] SuBtract with Carry from accumulator
[10] Accumulator Shift Left one bit
[11] Shift Right one bit
[12] Rotate One bit Left
[13] Rotate One bit Right

3.6.3 Branch instructions: BCC, BCS, BEQ, BMI, BNE, BPL, BVC and BVS

You can branch the program with: BCC[14], BCS[15], BEQ[16], BMI[17], BNE[18], BPL[19], BVC[20] and BVS[21].
eg.: See 'Compare instructions...' section!

3.6.4 BReaK instruction: BRK

You can interrupt a running program with: BRK[22].
Flags: BZ (B=1)
eg.:

```
A 5000 00       BRK
```

3.6.5 Compare instructions: CMP, CPX, CPY and BIT

You can compare values with each other: CMP[23], CPX[24] and CPY[25]. BIT is a test on bit level.
Flags: NZC (BIT: N=m7, V=m6, Z=a and m)
eg.:

```
A 5000 A2 00    LDX #$00
A 5002 AD FF 4F LDA #$4FFF
A 5005 9D 00 0C STA $0C00,X
A 5008 E8       INX
A 5009 EE FF 4F INC $4FFF
A 500C E0 00    CPX #$00
A 500E D0 F2    BNE $5002
```

[14]Branch on Carry Clear, jump when C=0
[15]Branch on Carry Set, jump when C=1
[16]Branch on zero (EQual), jump when Z=1
[17]Branch on result MInus, jump when N=1
[18]Branch on not zero (Not Equal), jump when Z=0
[19]Branch on result PLus, jump when N=0
[20]Branch on oVerflow Clear, jump when V=0
[21]Branch on oVerflow Set, jump when V=1
[22]BReaK the program
[23]CoMPare accumulator
[24]ComPare X register
[25]ComPare Y register

3.6.6 Flag instructions: CLC, CLD, CLI, CLV and SEC, SED, SEI

You can clear the bits of the Status Register: CLC[26], CLD[27], CLI[28], CLV[29] and set with: SEC[30], SED[31], SEI[32]
eg.: (make an interrupt: G5000(SYS20480)->'HELLO'; G502A(SYS20522)->end)

```
A 5000 78       SEI
A 5001 A9 00    LDA #$00
A 5003 A2 12    LDX #$12
A 5005 A0 50    LDY #$50
A 5007 8D 13 03 STA $0313
A 500A 8E 14 03 STX $0314
A 500D 8C 15 03 STY $0315
A 5010 58       CLI
A 5011 60       RTS
A 5012 A9 08    LDA #$08
A 5014 8D 00 0C STA $0C00
A 5017 A9 85    LDA #$85
A 5019 8D 01 0C STA $0C01
A 501C A9 8C    LDA #$8C
A 501E 8D 02 0C STA $0C02
A 5021 8D 03 0C STA $0C03
A 5024 A9 8F    LDA #$8F
A 5026 8D 04 0C STA $0C04
A 5029 40       RTI
```

```
A 502A 78       SEI
A 502B A9 00    LDA #$00
A 502D A2 0E    LDX #$0E
A 502F A0 CE    LDY #$CE
A 5031 8D 13 03 STA $0313
A 5034 8E 14 03 STX $0314
A 5037 8C 15 03 STY $0315
A 503A 58       CLI
A 503B 60       RTS
```

[26] CLear Carry flag, C=0
[27] CLear Decimal mode, D=0
[28] CLear Interrupt flag, I=1 (inverse)
[29] CLear oVerflow flag, V=0
[30] SEt Carry flag, C=1
[31] SEt Decimal mode, D=1
[32] SEt Interrupt flag, I=0 (inverse)

3.6.7 Increment / Decrement instructions: INC, INX, INY and DEC, DEX, DEY

You can increment a value of a register by 1 with: INC[33], INX[34], INY[35], and decrement by 1 with: DEC[36], DEX[37], DEY[38]
Flags: NZ
eg.: (XR:=XR+1; write '!' on the top-left of the screen)

```
A 5000 A2 0A    LDX #$0A
A 5002 E8       INX
A 5003 EE 00 0C INC $0C00
```

3.6.8 Jump instructions: JMP, JSR

You can make a direct jump to a memory address with: JMP[39], and call a subroutine: JSR[40].
eg.: (write 'AB' on the top-left of the screen)

```
A 5000 20 09 50 JSR $5009
A 5003 A9 02    LDA #$02
A 5005 8D 01 0C STA $0C01
A 5008 60       RTS
A 5009 A9 01    LDA #$01
A 500B 8D 00 0C STA $0C00
A 500E 60       RTS
```

[33] INCrement memory
[34] INcrement X register
[35] INcrement Y register
[36] DECrement memory
[37] DEcrement X register
[38] DEcrement Y register
[39] JuMP to a new location
[40] Jump to a SubRoutine

3.6.9 Loading instructions: LDA, LDX, LDY

You can loading a value into a register: LDA[41], LDX[42] and LDY[43].
Flags: NZ
eg.: (AR=10; XR=11; YR=12;)

```
A 5000 A9 0A    LDA #$0A
A 5002 A2 0B    LDX #$0B
A 5004 A0 0C    LDY #$0C
```

3.6.10 Logical instructions: AND, ORA, EOR

You can make logical operation on the bit level with AND[44], ORA[45] and EOR[46].
Flags: NZ
eg.: (AR=64+32)

```
A 5000 A9 40    LDA #$40
A 5002 09 20    ORA #$20
```

3.6.11 No OPeration instruction: NOP

It does nothing, only wastes the time: NOP[47].
eg.:

```
A 5000 EA       NOP
```

[41] LoaD Accumulator
[42] LoaD X register
[43] LoaD Y register
[44] "AND" logical operation
[45] "OR" logical operation
[46] "XOR" logical operation (EXclusive-OR)
[47] No OPeration

3.6.12 Return instructions: RTI, RTS

You can return from an interrupt with: RTI[48], and return from a subroutine: RTS[49].
eg.: See the example of the 'Flag instructions...' section!

3.6.13 Stack instructions: PHA, PHP, PLA, PLP

You can push a value to the stack with: PHA[50], PHP[51], and pull with: PLA[52], PLP[53].
eg.: (write 'A' on the top-left of the screen)

```
A 5000 A9 02    LDA #$01
A 5002 48       PHA
A 5003 A9 02    LDA #$02
A 5005 68       PLA
A 5006 8D 00 0C STA $0C00
```

3.6.14 Segment instruction: SEG

It enables you to set the using DATA[54] segment: SEG[55].
eg.: (setting DS to 01)

```
A 5000 FF 01    SEG #$01
```

[48] ReTurn from an Interrupt, NVBDIZC
[49] ReTurn from a Subroutine
[50] PusH Accumulator to the stack
[51] PusH Processor status to the stack
[52] PuLl Accumulator from the stack, NZ
[53] PuLl Processor status from the stack, NVBDIZC
[54] see at [0]:07FD and [0]:07FE
[55] SEGment

3.6.15 Storing instructions: STA, STX, STY

You can storing a value of a register into the memory: STA[56], STX[57] and STY[58].
eg.: ('ABC' on the top-left of the screen)

```
A 5000 A9 01    LDA #$01
A 5002 A2 02    LDX #$02
A 5004 A0 03    LDY #$03
A 5006 8D 00 0C STA $0C00
A 5009 8E 01 0C STX $0C01
A 500C 8C 02 0C STY $0C02
```

3.6.16 Transfer instructions: TAX, TAY, TXA, TYA, TSX, TXS

You can move a value of a register to another register: TAX[59], TAY[60], TXA[61], TYA[62], TSX[63] and TXS[64].
Flags: NZ (except for TXS)
eg.: (write 'B' on the top-left of the screen)

```
A 5000 A9 02    LDA #$02
A 5002 AA       TAX
A 5003 8E 00 0C STX $0C00
```

[56] STore Accumulator
[57] STore X register
[58] STore Y register
[59] Transfer Accumulator to X register
[60] Transfer Accumulator to Y register
[61] Transfer X register to Accumulator
[62] Transfer Y register to Accumulator
[63] Transfer Stack pointer to X register
[64] Transfer X register to Stack pointer

Chapter 4

Memory

The memory is divided into 256 segments. The size of a segment is 64 KiB, so the whole memory is 2^{24} = 16777216 bytes = 16384 KiB = 16 MiB (256*65536bytes). Each segment has got a function:

- 0 = System area
- 1-200 = BASIC area
  - 23-168 = Database area (3-PLUS-1)
  - 169-198 = Spreadsheet area (3-PLUS-1)
  - 199-200 = Word Processor area (3-PLUS-1)
- 201-250 = Variables
- 251-254 = TrueColor graphics screen (RGBA)
- 255 = 256colors graphics screen

Well, the **[segment]:offset** means a memory address where **segment** is a decimal number and **offset** is a hexadecimal number. For example: [0]:0C00 = the first character of the text screen.

When you start 3-PLUS-1 (sys0 or F1), the BASIC area is going to be changed after the 23rd segment until the 200th.

4.1 System area: [0]:0000-[0]:FFFF

This is the most complexed segment. It contains the significant foot-stone of the main system. Its size is 1 segment, so 65536 bytes.

4.1.1 Screen mode: [0]:0083

value	screen
0	full graphic, no text
1..12	text line number from bottom of the screen
13..24	text line number (-12) from top of the screen
25..44	text line number (-24) from left of the screen
45..64	text line number (-44) from right of the screen
65..74	text line number (-64) around the screen
75..	text window given from [0]:07E5 to [0]:07E8

Table 4.1: Screen mode

Figure 4.1: Share the screen: [0]:0083

4.1.2 ForeGround Color (for text): [0]:0085

It is a color index of the foreground color for text screens. For example: [0]:0085=14 –> the color of the foreground (eg.:text) is yellow.

4.1.3 BackGround Color (for text): [0]:0086

It is a color index of the background color for text screens. For example: [0]:0086=1 –> the color of the background is blue.

4.1.4 Timer: [0]:00A3 - [0]:00A5

TV4B has got a timer. [0]:00A5 is incrementing at every 100 milliseconds. [0]:00A4 is incrementing at every 256*100 milliseconds (about 25,6 seconds). [0]:00A3 is incrementing at every 256*256*100 milliseconds (about 109,2 minutes).

4.1.5 System stack: [0]:0100 - [0]:01FF

TV4B uses this area for pushing values to / pulling values from. This is a LIFO-memory. LIFO means Last-In-First-Out. What you push the last, that you pull the first... SP register shows you the top of the stack.

4.1.6 Loading character sets: [0]:053D

When you change CBM to PC8x8 at [0]:07FF!

value	character set
0.	PC charset
1.	COMPUTER charset
2.	CIRILL charset

Table 4.2: Character sets

4.1.7 The control byte: [0]:07FF

bit	function if 0	function if 1
0.	CBM charset	PC charset
1.	16 color text	256 color text
2.	normal text	inverse text
3.	temp for inverse	temp for inverse
4.	normal text	blink text
5.	temp for blink	temp for blink
6.	graphic off	graphic on
7.	320x200 bit/pixel	320x200 byte/pixel

Table 4.3: The control byte

4.1.8 Color attributes for 16c text screen: [0]:0800 - [0]:0BE7

It contains the color attributes for the 16 color text screen. Each byte of this area has got two part: the one is the upper four bit of the byte (this is the foreground color), the other part is the lower four bit of the byte (this is the background color).

4.1.9 Text screen: [0]:0C00 - [0]:0FE7

This is the default 16 color text screen. The text screen has got 40 columns and 25 rows of characters, so there are 1000 characters on the screen. For example, after you putting 1 value to the [0]:0C00 memory address, 'A' is appeared in the top-left corner of the screen.

You can set a dimension of the window where you type text from [0]:07E5 to [0]:07E8 –> Bottom, top, left, right.

4.1.10 Stack for iteration I.: [0]:3F40 - [0]:4340

TV4B uses this area for handling GOSUB-RETURN.

4.1.11 Stack for Lengyel's Form: [0]:14EA - [0]:15EA

TV4B uses the stack to make Lengyel's Form.

4.1.12 Queue for Lengyel's Form: [0]:15EB - [0]:16ED

TV4B uses the queue to make Lengyel's Form.

4.1.13 Lengyel's Form: [0]:16EC - [0]:17ED

It contains the Lengyel's Form of the last arithmetic or logical calculation.

4.1.14 FG Color table for 256c graphic: [0]:1800 - [0]:1DE7

Each byte of this area is a color index of the foreground color of every characters on the graphic screen. Each character on the graphic screen is 8x8 pixels, so the pixels in this matrix are the same color.

4.1.15 BG Color table for 256c graphic: [0]:1C00 - [0]:1FE7

Each byte of this area is a color index of the background color of every characters on the graphic screen. Each character on the graphic screen is 8x8 pixels, so the pixels in this matrix are the same color.

4.1.16 Graphics screen: [0]:2000 - [0]:3F3F

This is the default high resolution graphics screen. It is a bitmap (bit / pixel) area. Each bit of this area is according to the appropriate pixel of the screen.

For example, it is easy to put a bitmap of a character to it. After you putting 129d (81h) value to 2000h, you can see 10000001b as pixels on the screen.

4.1.17 Stack for iteration II.: [0]:3F40 - [0]:4340

TV4B uses this area for handling values after TO.

4.1.18 Stack for iteration III.: [0]:4341 - [0]:4741

TV4B uses this area for handling values after STEP.

4.1.19 Stack for iteration IV.: [0]:4742 - [0]:4C42

TV4B uses this area for handling FOR-NEXT variables.

4.1.20 Screen refreshing attributes: [0]:4C43 - [0]:4CBF

TV4B uses this area for refreshing the appropriate 8x8 part of the screen.

4.1.21 Unused area: [0]:4CC0 - [0]:CDA9

You can use this area for anything you want. The size of it 33002 bytes.

4.1.22 Secret code: [0]:CDAA - [0]:CE07

This is an Easter Egg for storing the address, e-mail and phone number of the author. Its size is 94 bytes.

4.1.23 Interrupt: [0]:CE0E

TV4B has got an interrupt on a separated thread which runs every 1/100 minute[1] (if you set the Interrupt-flag: see the example in 'Flag instructions...' section!).

You can set the start address of the desired code at [0]:0313, [0]:0314 and [0]:0315. You must return from the interrupt-code with a RTI instruction.

4.1.24 Character set (CBM): [0]:D000 - [0]:D7FF

This is the default character set to view the text on the screen. It is called Commodore Business Machine character set included the Hungarian accentuated letters.

The character set consists of the bitmaps of the characters. Each bitmap is a 8x8 matrix (8 bytes). So 256*8 bytes = 2048 bytes.

4.1.25 Character set (PC): [0]:D800 - [0]:DFFF

This is the other character set to view the text on the screen. It is called Personal Computer character set included the Hungarian accentuated letters.

The character set consists of the bitmaps of the characters. Each bitmap is a 8x8 matrix (8 bytes). So 256*8 bytes = 2048 bytes.

[1] [0]:00CF shows the running interrupt

4.1.26 FG Color table for 256c text: [0]:E000 - [0]:E3E7

Each byte of this area is a color index of the foreground color of every characters on the text screen.

4.1.27 BG Color table for 256c text: [0]:E400 - [0]:E7E7

Each byte of this area is a color index of the background color of every characters on the text screen.

4.1.28 Inverse: [0]:E800 - [0]:E87E

It is a bitmap area for printing characters in inverse-mode on the text screen. There are 1000 characters on the screen, because 40x25 is 1000 and 1000 bit / 8 = 125 bytes. If you set one bit to 1, the appropriate character will be viewed in inverse-mode until setting it to zero.

4.1.29 Frame Color: [0]:E87D

It is a color index of the frame color. For example: [0]:E87D=4 –> the color of the frame is red.

4.1.30 ForeGround Color (for graphic): [0]:E87E

It is a color index of the foreground color for graphics screens. For example: [0]:E87E=14 –> the color of the foreground (eg.:text) is yellow.

4.1.31 BackGround Color (for graphic): [0]:E87F

It is a color index of the background color for graphics screens. For example: [0]:E87F=1 –> the color of the background is blue.

4.1.32 Blink: [0]:E880 - [0]:E8FE

It is a bitmap area for blinking characters of the text screen. There are 1000 characters on the screen, because 40x25 is 1000 and 1000 bit / 8 = 125 bytes. If you set one bit to 1, the appropriate character will be blinking until setting it to zero.

4.1.33 Sections: [0]:E8FD - [0]:E916

There are 25 sections on the text screen at the beginning.

4.1.34 User typed text: [0]:E917 - [0]:ECFF

When you press ENTER, the text in the current section is stored in this area.

4.1.35 Interpreter area: [0]:ED00 - [0]:FCFF

TV4B uses this area to compile the commands or instructions. The first two bytes is the length of machine code.

4.1.36 Input/Output area: [0]:FD00 - [0]:FEFD

TV4B uses this area to handle input/output devices. For example: keyboard.

FD30	**FE**	**FD**	**FB**	**F7**	**EF**	**DF**	**BF**	**7F**
FE	DEL	RET	Ł	F4	F1	F2	F3	@
FD	3	W	A	4	Z	S	E	SHF
FB	5	R	D	6	C	F	T	X
F7	7	Y	G	8	B	H	U	V
EF	9	I	J	0	M	K	O	N
DF	DWN	P	L	UP	.	.	-	,
BF	LFT	*	,	RHT	INS	7	+	,
7F	1	HME	CTR	2	SPC	ALT	Q	TAB

Table 4.4: Keyboard scan codes

eg.: waiting for pressing H on the keyboard

```
.5000 A9 F7    LDA #$F7
.5002 8D 30 FD STA $FD30
.5005 A9 00    LDA #$00
.5007 8D 08 FF STA $FF08
.500A AD 08 FF LDA $FF08
.500D C9 DF    CMP #$DF
.500F D0 F4    BNE $5005
.5011 60       RTS
```

4.1.37 TED and microcodes: [0]:FF00 - [0]:FFFF

TED[2] is one of the most interesting tool for manipulating the computer. When you write a byte to this area, the system does something, such as making sound, changing video mode, setting colors, handling keyboard and joystick, etc.

Register	**function**
FF00	The first counter (low byte)
FF01	The first counter (high byte)
FF02	The second counter (low byte)
FF03	The second counter (high byte)
FF04	The third counter (low byte)
FF05	The third counter (high byte)
FF06	Screen mode#1
FF07	Screen mode#2
FF08	Keyboard or joystick
FF09	IRQ
FF0A	IRQ enabled
FF0C	Cursor position#2
FF0D	Cursor position#1
FF0E	Frequency for sound generator#1 (low byte)
FF0F	Frequency for sound generator#2 (low byte)
FF10	Frequency for sound generator#2 (high byte)
FF11	Sound generators
FF12	Frequency for sound generator#1 (high byte)
FF13	Character table address
FF14	Graphical screen address
FF15	Background color for text
FF16	Foreground color for text
FF17	Background color for graphic
FF18	Foreground color for graphic
FF19	Frame color
FF1A	Special effects
FF3E	Segment of character table
FF3F	Segment of graphic screen

Table 4.5: TED registers

See more details about TED registers in the memory map...

[2] **T**ext **E**diting **D**evice

Microcodes are mini procedures for different functions. All available micro codes is a RTS. You can use them 'a jump to it' with SYS (in BASIC) or JSR (in Assembly).

Microcode	**function**
FFD2	Print the value of AR to the text screen
FFF6	Soft reset (F11)
FFFE	Reboot the computer
FFFF	Shutdown the computer

Table 4.6: Microcodes

4.2 BASIC area: [1]:0000-[200]:FFFF

It contains your BASIC program. You can modify the start address of the BASIC area at [0]:002A, [0]:002B and [0]:002C. Its size is 200 segment, so 200*65536 bytes (12800 KiB = 12,5 MiB).

4.3 3-PLUS-1: [23]:0000-[200]:FFFF

The BASIC area is going to be changed when you run the embedded programs also known as 3-PLUS-1. This 3+1 application will use this area for its works. The Database is between [23]:0000 and [168]:FFFF, the Spreadsheet is between [169]:0000 and [198]:FFFF and the Word Processor is between [199]:0000 and [200]:FFFF. The Graph don't use any amount of space in memory.

The Word Processor enables you to store 1000-line-length text, but you can merge more files, too. The Spreadsheet enables you to make calculations in 25 000 cells. In the end you can store 10 000 records in the Database application. More details in 3-PLUS-1 chapter and in appendix!

4.4 Variables: [201]:0000-[250]:FFFF

It contains your variables. This area is cleared each time of running a BASIC program or when you executed a CLR or NEW command / instruction. Its size is 50 segment, so 50*65536 bytes (3200 KiB = 3,125 MiB).

4.5 TrueColor: [251]:0000-[254]:FFFF

It contains the TrueColor graphics screen[3]. These four segment stands for RGBA. The segment of 251 ([251]:0000 - [251]:F9FF[4]) is the RED component for each pixel of the screen. The segment of 252 ([252]:0000 - [252]:F9FF) is the GREEN component for each pixel of the screen. The segment of 253 ([253]:0000 - [253]:F9FF) is the BLUE component for each pixel of the screen. The segment of 254 ([254]:0000 - [254]:F9FF) is the ALPHA component for each pixel of the screen.

4.6 256color: [255]:0000-[255]:FFFF

It contains the 256color graphics screen. Each byte of it ([255]:0000 - [255]:F9FF) is a color index which points to the Color Index Table.

4.7 Color Index Table: [255]:FA00-[255]:FDFF

It contains the RGBA components[5] of the 256 colors.

4.8 Mouse shape: [255]:FE00-[255]:FEFF

This 256 bytes are the mouse[6] cursor shape (16x16) given by color indexes. It is for text screen, bit/pixel and byte/pixel graphics screens. You can create the mouse cursor shape for TrueColor at [251]:FA00 (Red), [252]:FA00 (Green), [253]:FA00 (Blue) and [254]:FA00 (Alpha).

[3] when the graphic on and [0]:0084 > 0
[4] F9FFh + 1 = 64000d (320x200)
[5] 4x256 bytes = Red Green Blue Alpha for each color
[6] see from [0]:07EC to [0]:07F1 (mouse state)

Chapter 5

A little motivation

What is the meaning of all these? What is the aim of the Life? :-) Well, TV4B brings back of my happy childhood when I was programming all day free from care... Here is some example games (from the book: "Csupa Szuperjáték") to show you what I mean... After a lot of thinking about an idea and planing characters in order to come true your dreams, you could make a little program wich expresses your feelings and showing other people what the power of your mind!

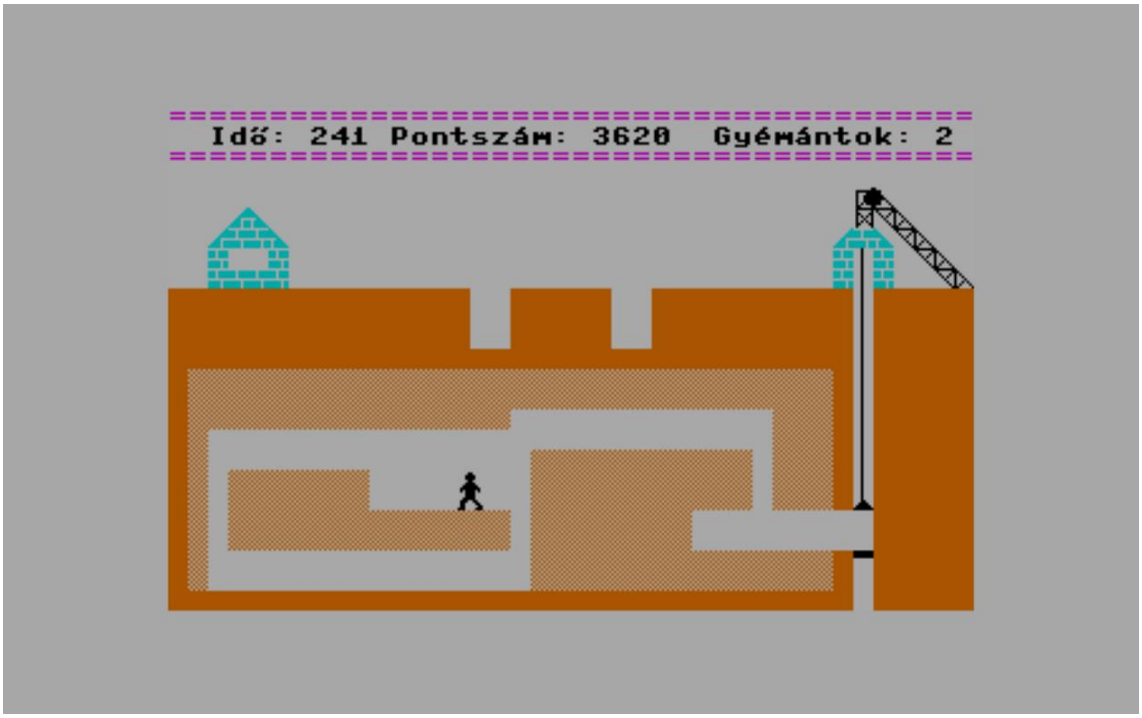

Figure 5.1: Gyémántbányász = Goldminer

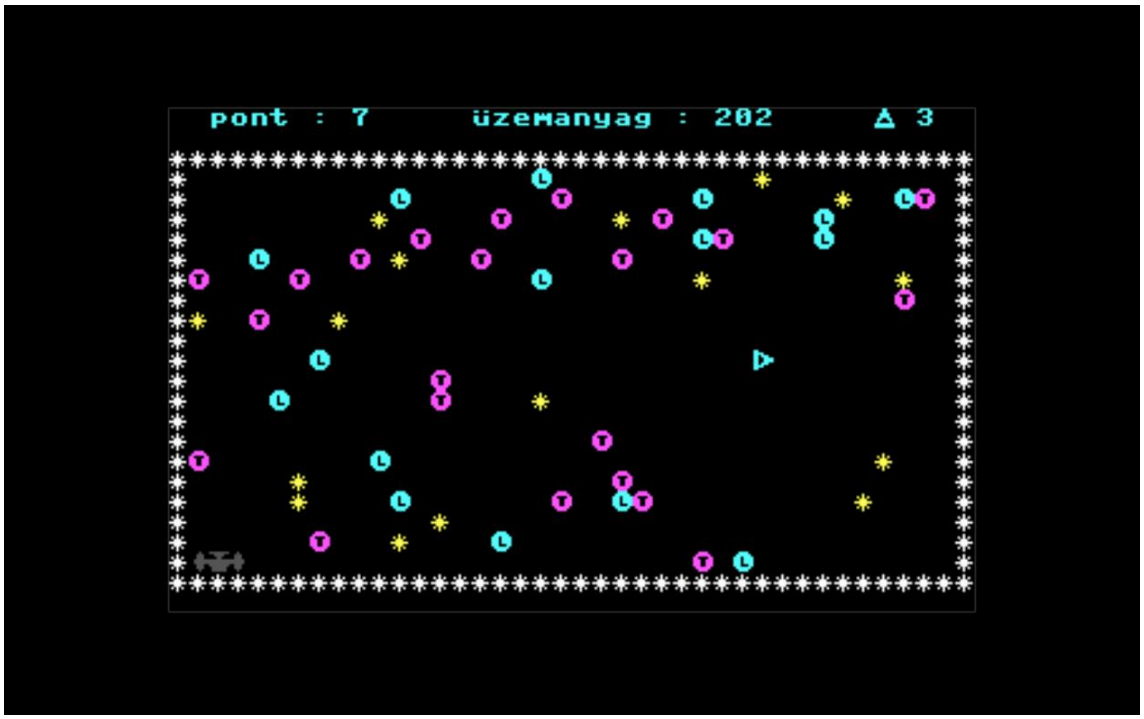

Figure 5.2: Űrkaland = Skyadventure

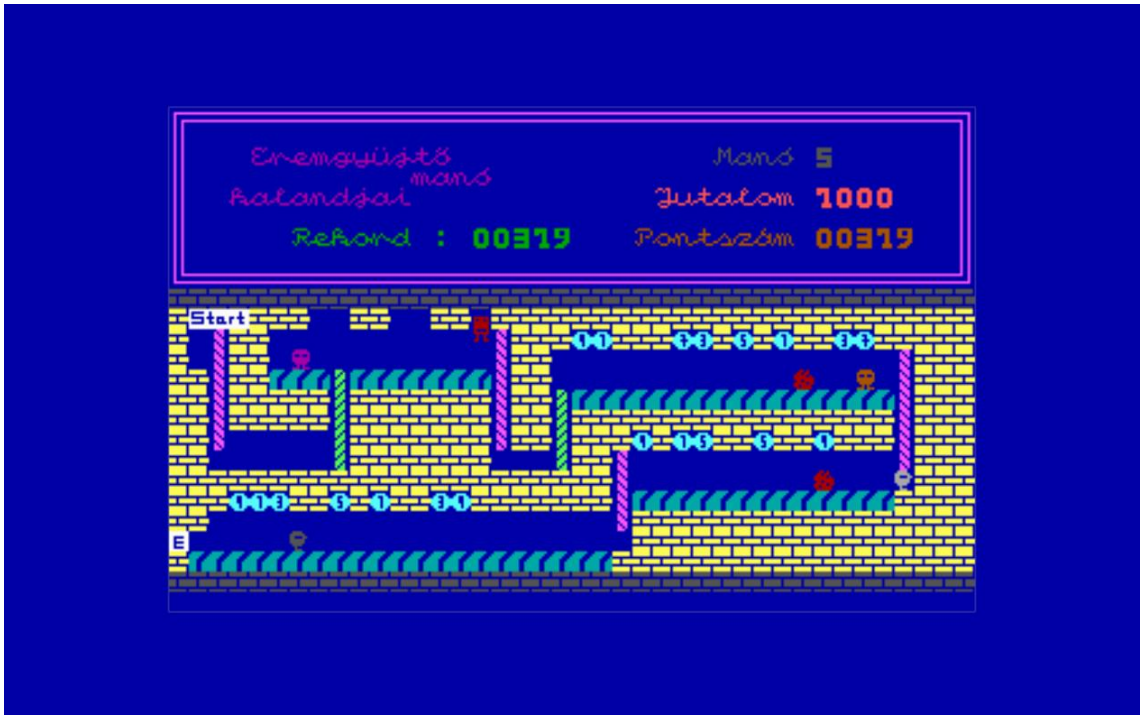

Figure 5.3: Éremgyűjtő manó kalandjai = Coincollector elf's adventures

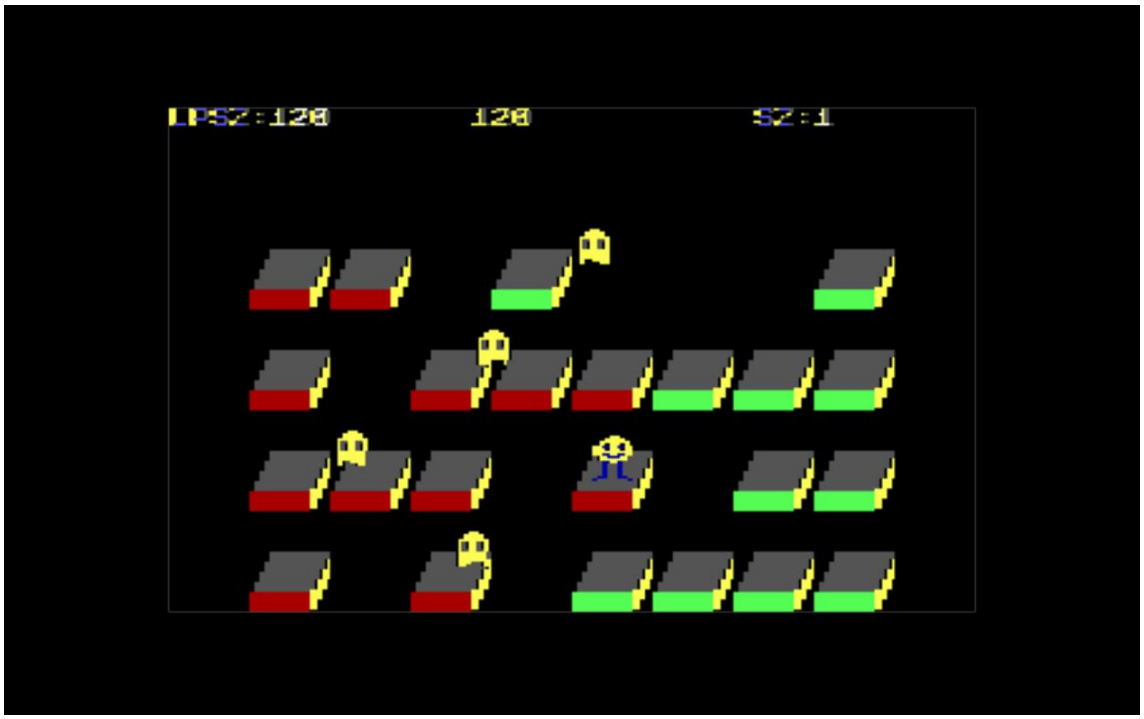

Figure 5.4: Kockafestő = Boxpainter

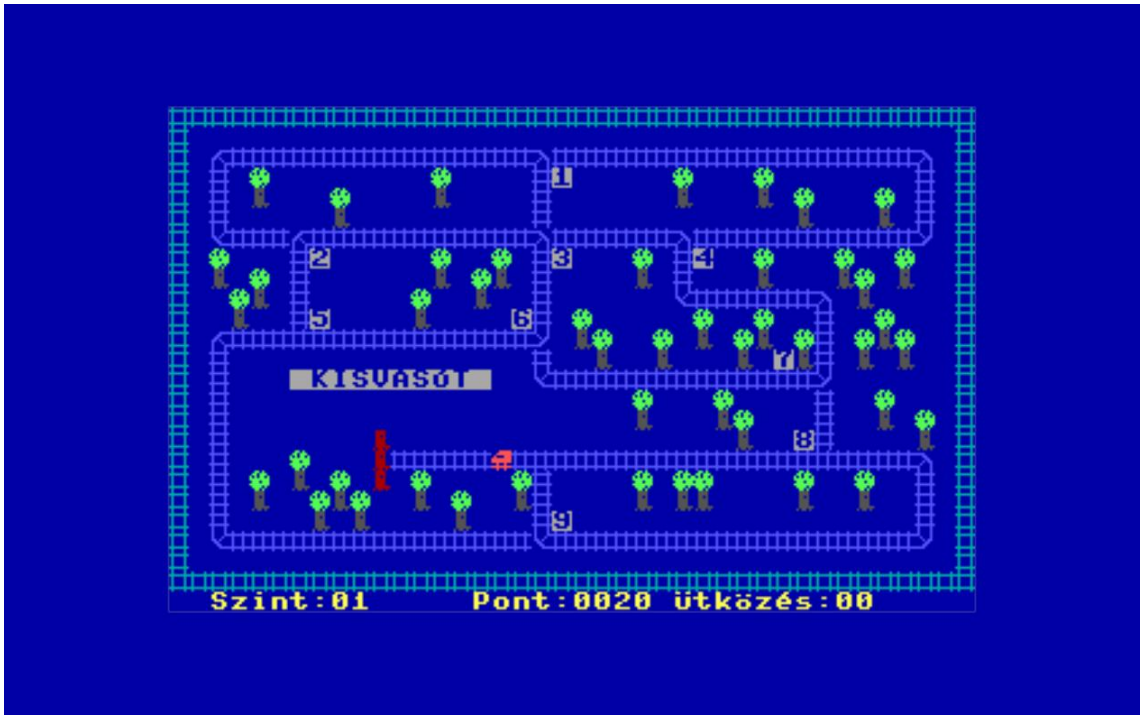

Figure 5.5: Kisvasút = Littletrain

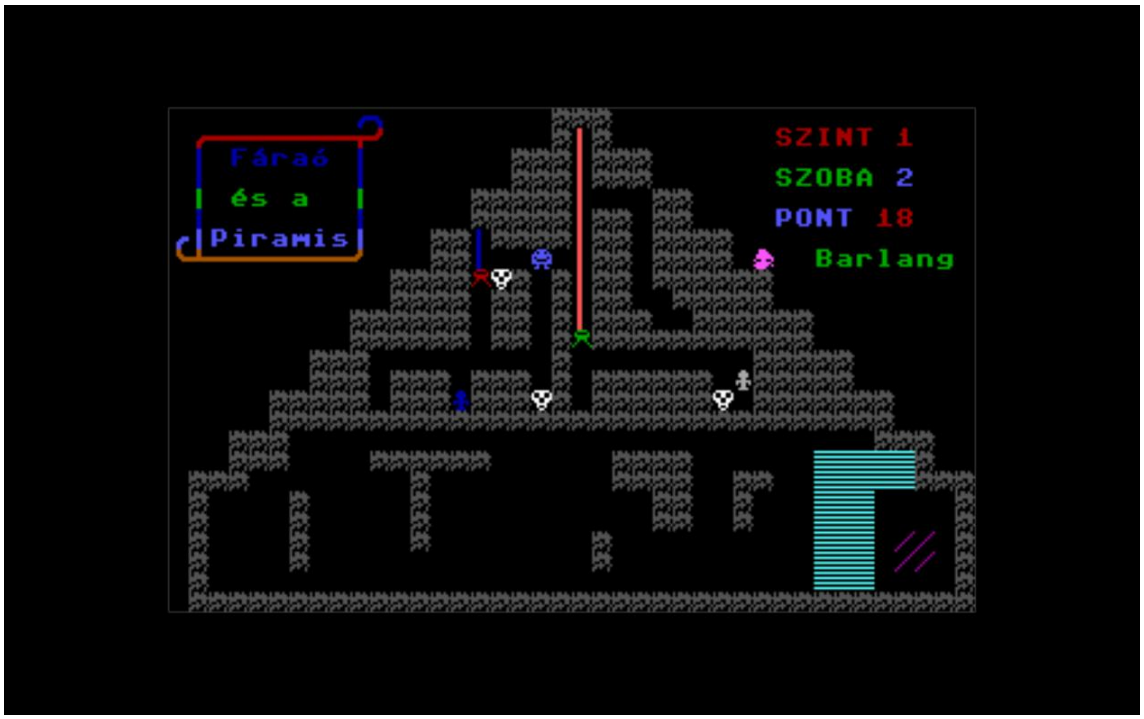

Figure 5.6: Fáraó és a piramis = Pharaoh and pyramid

Figure 5.7: Bolondos golyó = Fury ball

Chapter 6

Turbo Vision Disk

Turbo Vision Disk (TVD) is a virtual disk format for TV4B system. It consists of one or more (up to 1024) files. TVD enables you to store your project files in a single file.

The more files in a TVD, the more slower the writing operation in a TVD. Some functions do not work in a TVD, for instance: playing an MP3. What is the meaning of TVD then? You can store a plenty of little files in a TVD, so they allocate even less storage capacity.

The DIRECTORY, SAVE, LOAD and VERIFY Basic commands are also running normaly in TVD. As well as L, S and V in Assembly, too.

In that case you would like to protect your source codes from others, you can encode a TVD with the SAVE command. An encoded TVD is not able to be modified.

6.1 CHDIR

To change a TVD directory:
CHDIR<"tvd:dirname">

dirname the name of the TVD directory to change

eg.:
CHDIR"tvd:mixer.tvd": to change TVD directory to "mixer"

6.2 COPY

To copy a file to or from a TVD directory:
COPY<"tvd:filename1"> TO <"filename2">

filename1 the name of the file to copy
filename2 the name of the TVD file or a new file

eg.:
COPY"tvd:mixer.tv4" TO "mixer.tvd" : to copy "mixer.tv4" to mixer TVD
COPY"tvd:mixer.tv4" TO "mixer2.tv4" : to copy "mixer.tv4" from mixer TVD (if you are in)

6.3 DIRECTORY

It helps you to list the stored programs in a TVD.
DIRECTORY[<num>]

num page number

eg.:
DIRECTORY : list all files from the TVD
DIRECTORY2 : list all files (the 2nd page)

6.4 LOAD

LOAD command loads an encoded TVD from disk to the memory. It loads a file (you can give it with SAVE command beforehand) from TVD.
LOAD<"tvd:name">

name the name of the encoded TVD to load

eg.:
LOAD"tvd:program.tvd" : load encoded "program.tvd" and run

6.5 MKDIR

To make a TVD directory:
MKDIR<"tvd:dirname">

dirname the name of the TVD directory to make

eg.:
MKDIR"tvd:mixer.tvd": to make "mixer" TVD directory

6.6 RENAME

To rename a file in a TVD:
RENAME<"tvd:oldfilename"> TO <"newfilename">

oldfilename the name of the file to rename
newfilename the name of the new file

eg.:
RENAME"tvd:mixer.tv4" TO "mixing.tv4" : to rename "mixer.tv4"

6.7 SAVE

SAVE command encodes a TVD disk.
SAVE<"tvd:name">[,<num>]

name	the name of the TVD to encode
num	file number (0: the 1st; 1: the 2nd...)

eg.:
SAVE"tvd:program.tvd" : encode the program TVD (LOAD the first file(0th))
SAVE"tvd:program.tvd",5 : encoding, start with (D)LOAD the sixth file(5th)

6.8 SCRATCH

To remove a file from the current TVD directory:
SCRATCH<"tvd:filename">

filename	the name of the file to delete

eg.:
SCRATCH"tvd:mixer.tv4" : to delete the "mixer.tv4" file

Chapter 7

Utilities

In the course of years some applications are born in order to help TV4B.

- **Browser** - Read Me
- **DrawChar** - Char Designer
- **Prg2Tv4** - PRG converter
- **TVDC** - Turbo Vision Disk Commander
- **Newsreel** - Electronic newspaper

7.1 Browser

This is a "Read Me" utility for reading descriptions of TV4B programs.

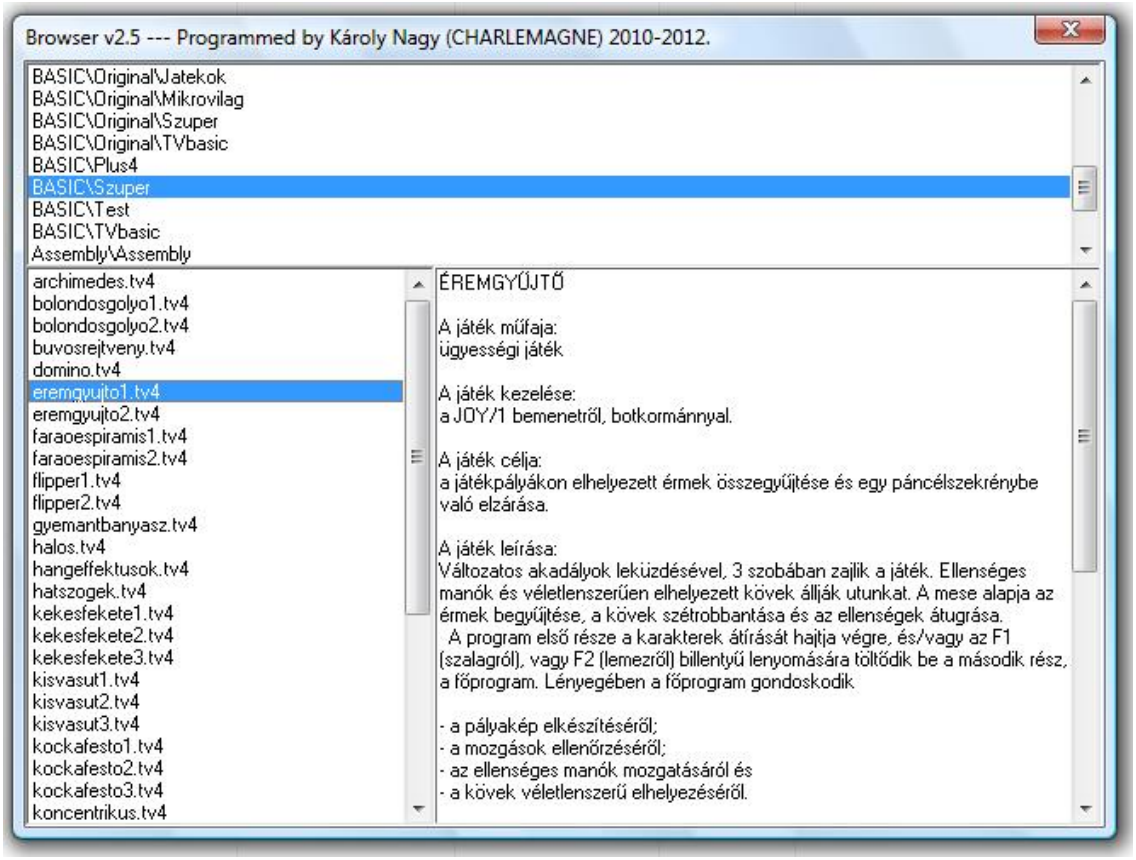

Figure 7.1: Browser - Read Me

It enables you to read about programs (such as the story, directions, running instructions, locations) before loading and running them. Browser is easy to use application, well you have only one task: keep clicking!

Utility –> Browser

7.2 DrawChar

This is a "Char Designer" utility for creating new characters or building up bigger sprites.

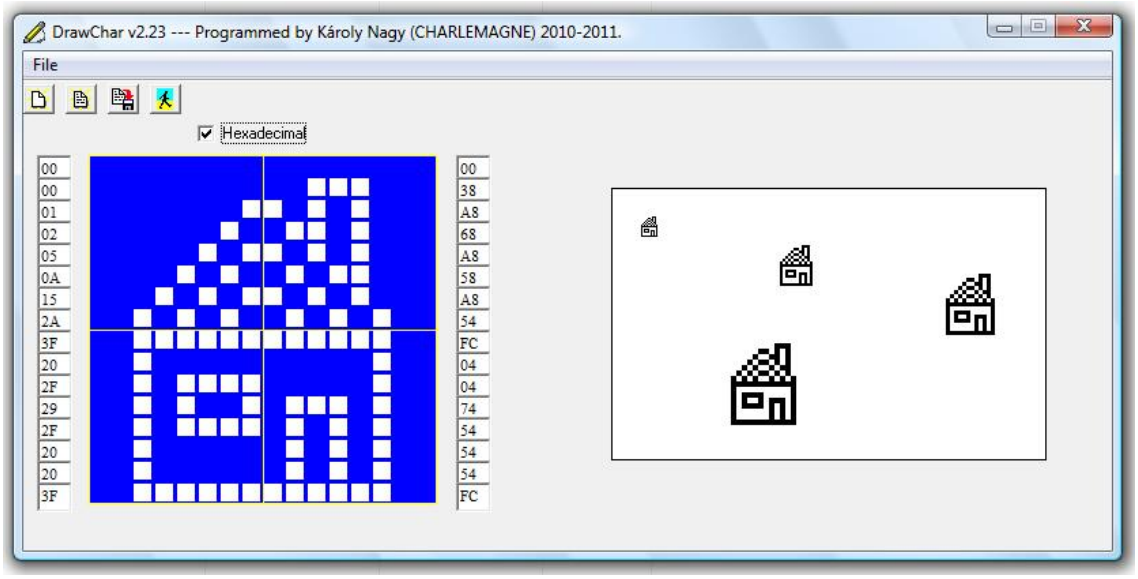

Figure 7.2: DrawChar - Char Designer

You can put down a pixel with the left mouse button, and take up with the right mouse button. DrawChar enables you to give numbers (decimal and hexadecimal) to draw a line of a character. It is good for demonstrating how the computer stores the characters in its memory.

If you have finished a character, you can save it on the storage. In that case you want to continue a previous work, you can load it from the storage.

Utility –> DrawChar

7.3 Prg2Tv4

This is a "PRG Converter" utility for converting PRG files to TV4.

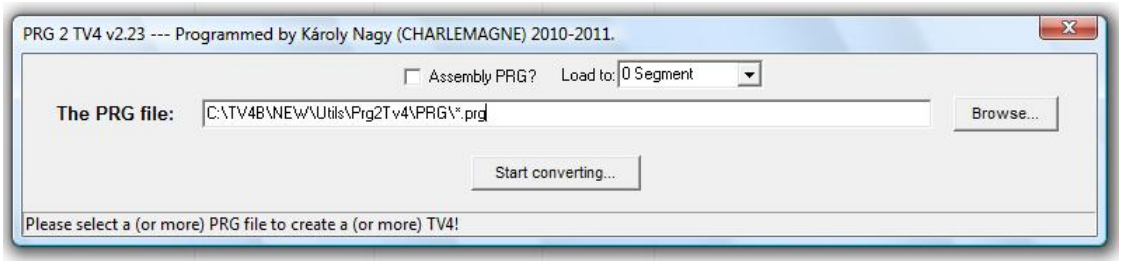

Figure 7.3: Prg2Tv4 - PRG Converter

PRG files are program files of the original Commodore computers. TV4B cannot load PRG in directly, because TV4B for example distinguishes lower case letters from upper case letters in the Basic programs.

You can convert one or more files off the reel with the help of Joker-characters (*, ?).

Utility –> Prg2Tv4

7.4 TVDC

This is a "Turbo Vision Disk Commander" utility for managing files in TVD virtual disks.

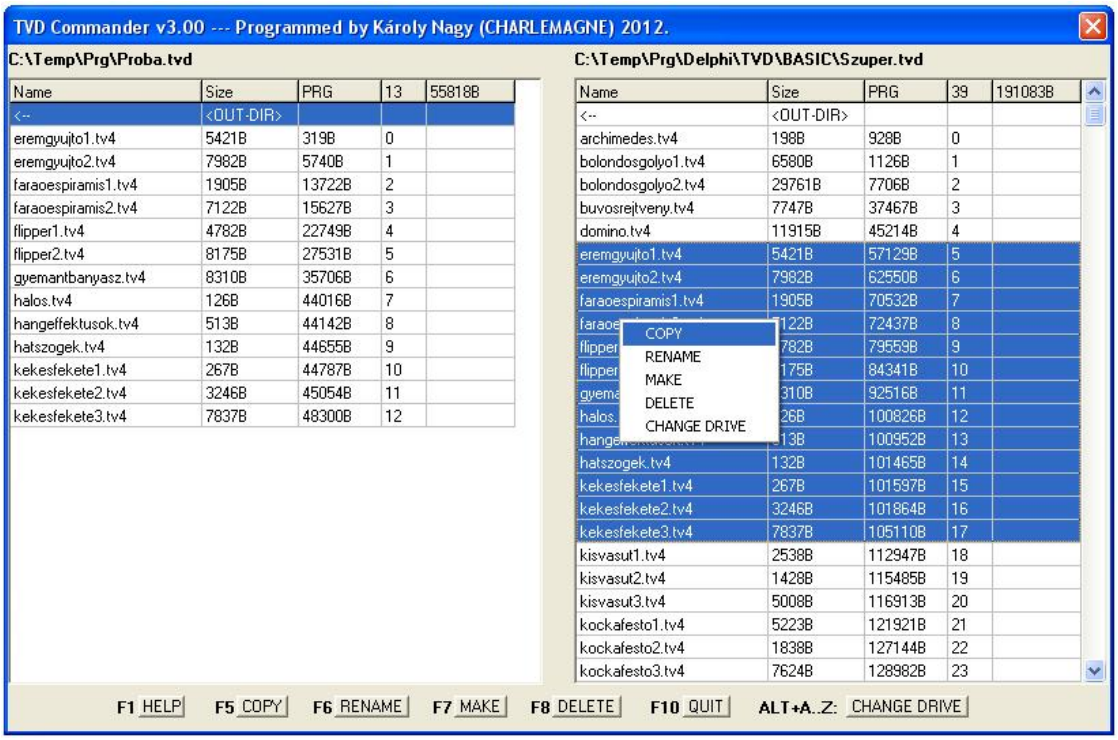

Figure 7.4: TVDC - Turbo Vision Disk Commander

Key	**function**
F5	Copy a (or more) file(s) to a TVD Copy a (or more) file(s) from a TVD Copy a (or more) file(s) from a TVD to in another TVD
F6	Rename a file in a TVD
F7	Make a TVD "directory"
F8	Delete a (or more) TVD disk(s) Delete a (or more) file(s) from a TVD
F10	Quit Exit
ALT+A..Z	Change drive
CTRL+A	Select all files

Utility –> TVDC

7.5 Newsreel

This is an "Electronic newspaper" utility for reading news with pictures and TV4B sourcecodes.

Figure 7.5: Newsreel - Electronic newspaper

By the way, you can also write a newspaper like that old yellow pages. If you use "picture" parameter, you are able to insert pictures at the end of your news. In that case the name of the picture[1] must be "picture.bmp" and the name of the newsreel is to "newsreel.news".

newsreel.exe picture

[1] a 24bpp BMP

command	function
<newsreel yyyymmddhhmmss>	ID for a newspaper yyyy: year mm: month dd: day hh: hour mm: minute ss: second
<headline>	8x16 PC chars and 3x size
<chapter>	8x8 PC chars and 3x size
<newspaper>	8x16 PC chars and 2x size
<sourcecode>	8x8 CBM chars and 2x size
<picture WxH S P>	Picture W: Width of the picture H: Height of the picture S: Size of a pixel P: Position of the picture (left,center,right)

Here comes a little example:

```
<newsreel 201205212121>
<headline>
Welcome to everybody!

<newspaper>
This is the first issue of my newspaper.

<chapter>
Hello World!

<sourcecode>
10 SCNCLR
20 PRINT"News v1.0"
30 GETKEY A$
```

Utility –> Newsreel

Chapter 8

3-PLUS-1

By the way TV4B is not only a BASIC and Assembly interpreter. What's more... After the date turning over 2014, I made up my mind to implement the 4 embedded programs of the original Commodore Plus/4 computer.

- One of these applications is Word Processor (1).
- The other one is Spreadsheet (2).
- The third one is Database (3).
- Finally the last one is Graph (+1). It's also known as Chart.

Word Processor, Spreadsheet and Database are complete applications. Graph is just an extension of Spreadsheet.

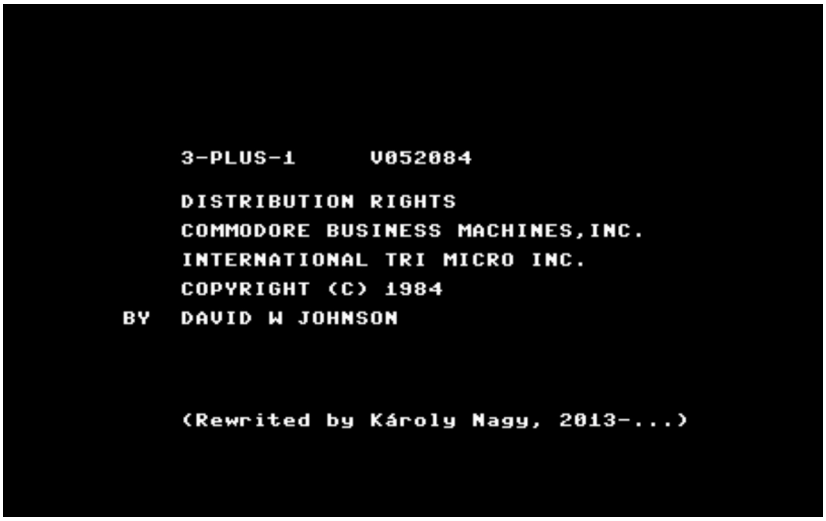

Figure 8.1: 3-PLUS-1

You can enter to command line mode with (right)CTRL+C.

8.1 Word Processor

Word Processor enables you to type, edit and print a text. The dimension of the text area is 80x1000. Well there are 80 columns and 1000 lines. These properties are shown as L= and C= at the bottom of the screen.

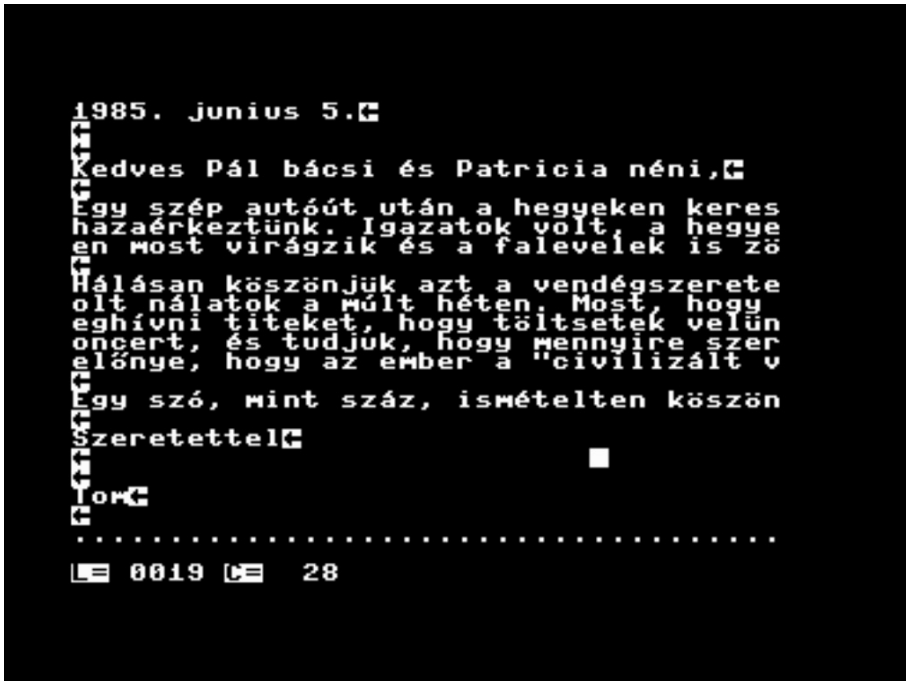

Figure 8.2: Word Processor

So the cursor is in the 19th line and the 28th column. You don't see the whole text area, you are able to see only 22 lines and 37 columns of it. When you reach the edge of the screen, it will be scrolling to the appropriate direction.

You can set tabulators and pointers. You can also create, delete and copy blocks like the clipboard on PC. The size of the clipboard is 484 lines.

8.1.1 Keys

Key	Explanation
F1	Jump to the next line (at the first column)
F2	Jump to the next line (at the 41st column)
END	Go to the last (1000th) line
ENTER	Go to next line with a break
HOME	Go to the first line
TAB	Jump to the next tabulator
(left) CTRL+9	Switch to inverse: executing command while printing
(left) CTRL+0	Turn off inverse: end of executing command
(right) CTRL+C	Enter to command line
(right) CTRL+Q	Repeating the last command
(right) CTRL+TAB	Set a tabulator
(right) SHIFT+ENTER	Go to next line without break

8.1.2 Commands

When you press (right)CTRL+C, you can type the following command:

Command	Explanation
ca	**CA**talogue list of saved documents
cb	**C**reate **B**lock save the text to the clipboard until the nearest pointer
cm	**C**lear **M**emory clear the text from the memory
cp	**C**lear **P**ointers clear all pointers
ct	**C**lear **T**abulator clear the current tabulator
db	**D**elete **B**lock delete the text to the nearest pointer
df	**D**elete **F**ile delete a document
dl	**D**elete **L**ine delete the current line
ep	**E**rase **P**ointer erase the current pointer
ib	**I**nsert **B**lock paste the text from the clipboard
il	**I**nsert **L**ine insert an empty line
lf	**L**oad **F**ile load a document
mf	**M**erge **F**ile merge another document
poke o,v	**POKE** set a value of [0] segment
re	**RE**change search and change texts
sf	**S**ave **F**ile save the document

Command	**Explanation**
sp	**S**et **P**ointer set a pointer
sr	**S**ea**R**ch search a text
tc	**T**o our **C**omputational switch to Spreadsheet
tf	**T**o our **F**ilemanager switch to Database
*p	all **P**rint print the text (100 lines per page)

8.1.3 Printing Commands

If you want to set the text properties, enter inverse-mode with (left)CTRL+9 and type a command. When you finished it, you have to come out of inverse-mode with (left)CTRL+0. These commands work only upon printing!

Printing command	Explanation
#page;	number pages
#rc;	write the record number
center;	align text to the center
eof?;	increase the record number
fld n;	write the n-th field
justify;	align text in justify
left;	align text to the left
linkfile "filename";	load and print another document
lmarg n;	set the left margin
nextpage;	start a new page
no#page;	stop to number pages
nojustify;	align text to the left
nowrap;	no word wrap
pagelen n;	set n lines per pages
rc n;	set the n-th record
right;	align text to the right
rmarg n;	set the right margin
set#pg n;	set page number to n
ttl n;	write the name of the n-th field
wrapon;	word wrap

8.1.4 Circular letter

This chapter shows you how to use the Word Processor with the Database. There are 2 records in the database:

- 1. record 1. field (Név): Nagy Károly
- 1. record 2. field (Cím): Pálmonostora
- 2. record 1. field (Név): Charlie Bebit
- 2. record 2. field (Cím): Kiskunfélegyháza

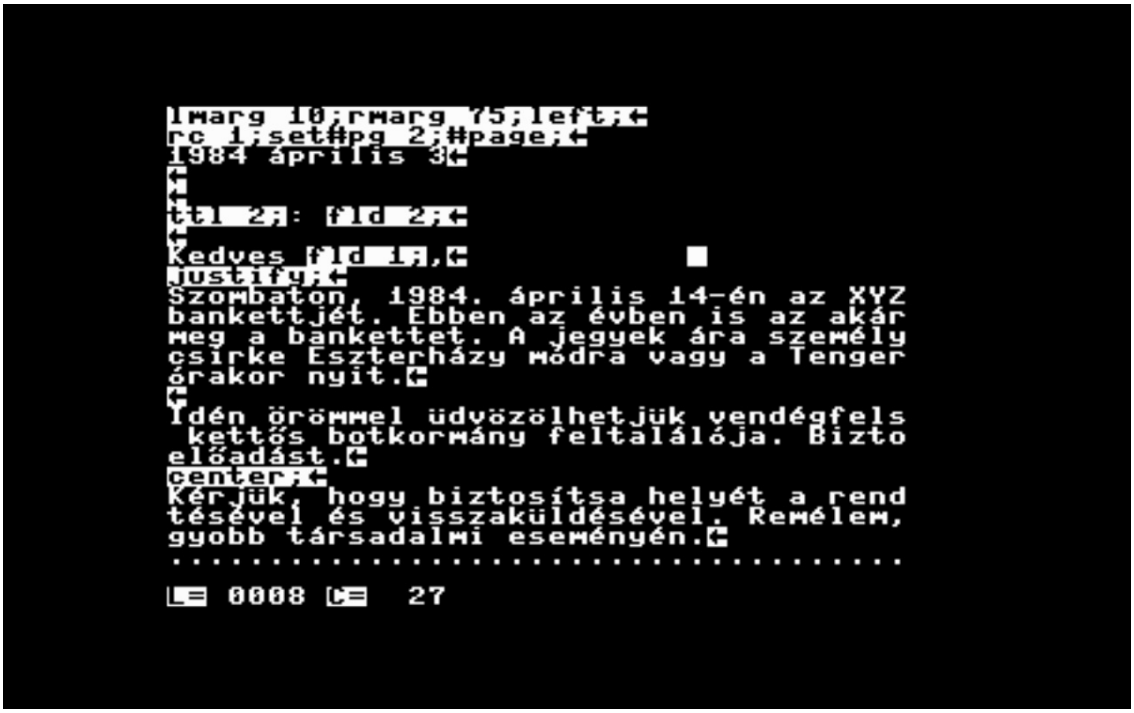

Figure 8.3: Circular letter

Well, for the first time you have to create a database that consists of some records for people datas. Then you have to write a pattern letter with printing commands (like in the picture above: `ttl` for title of a field, `fld` for value of a field, etc.). It's very important to write an `eof?;` command at the end of the circular letter. This command increases the record number until reaching the last one.

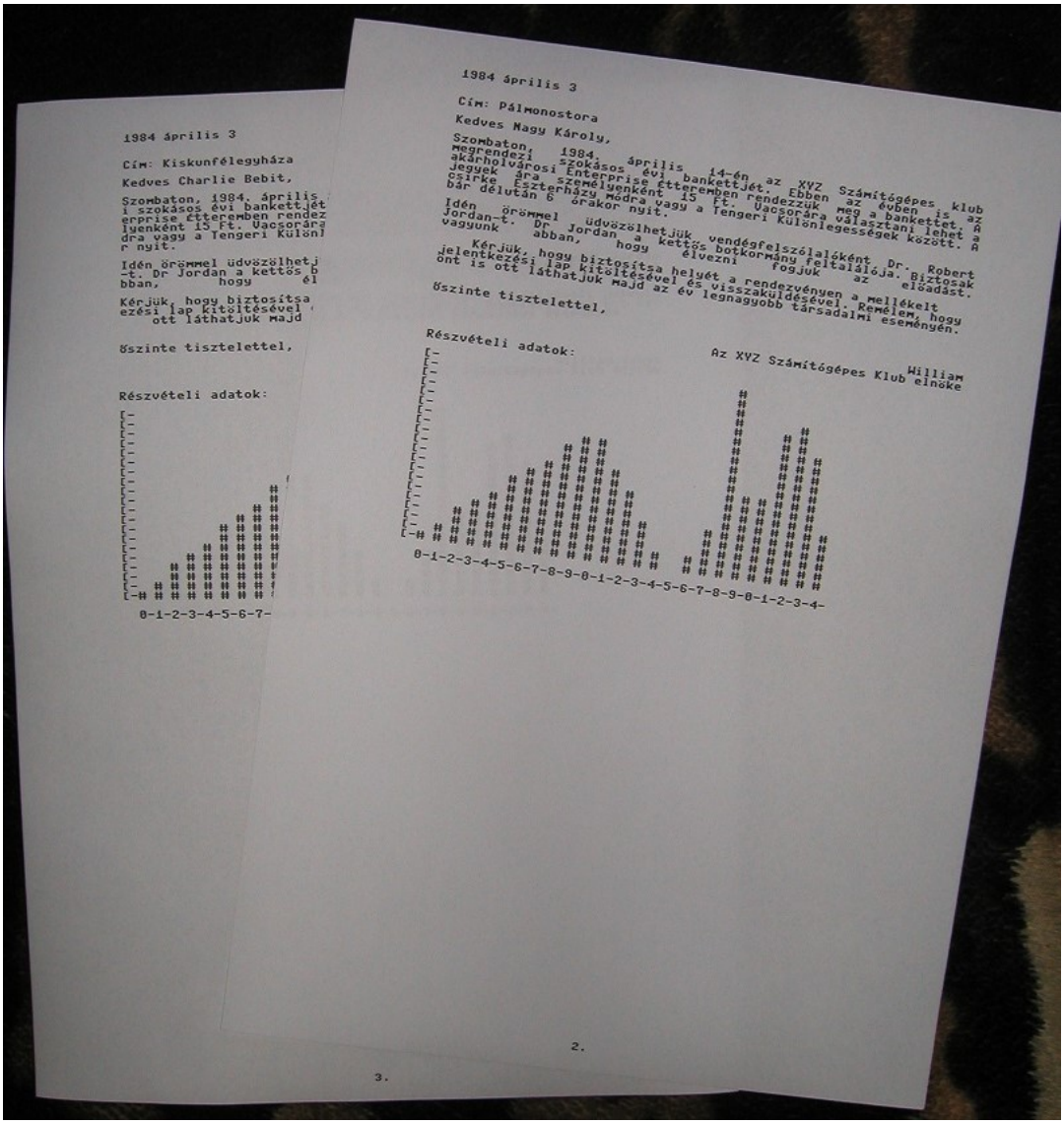

Figure 8.4: Printed circular letter

Well, as you can see the printing commands don't appear on the printed pages, of course. It's worth typing `nowrap;` command before making a graph in Spreadsheet! The record and page number increase automatically.

8.2 Spreadsheet

Spreadsheet enables you to make calculations. The dimension of the sheet area is 25x1000. There are 25 columns and 1000 rows. These properties are shown as R;C at the bottom of the screen. Columns are signed by C and a number such as C 1, C 2, C 3, ... Rows are signed by R and a number such as R 1, R 2, R 3, ... The junction of a column and a row is a cell. You can refer to a cell with its row and column number separated by a semicolon. A cell can consist a number, a text or a formula. Numbers are floating point numbers in the range of -33554431 and 33554431. The length of a text or a formula can be up to 36 character long.

Figure 8.5: Spreadsheet

The active cell is in the junction of the 12th row and the 2nd column. It's worth using this kind of reference whenever you want to make calculation. This is the main point of a Spreadsheet application.

You can move up and down the cursor with up and down arrow. You can move left and right the cursor with F1 and F2.

8.2.1 Keys

Key	Explanation
F1	Move left the cursor
F2	Move right the cursor
(left) ALT+-	↑ (power)
(left) ALT+O'	← (store)
(right) CTRL+C	Enter to command line
(right) CTRL+F	Switch to formula-mode
(right) CTRL+L	Align a cell content to the left
(right) CTRL+N	Switch to number-mode
(right) CTRL+Q	Repeating the last command
(right) CTRL+R	Align a cell content to the right
(right) CTRL+T	Switch to text-mode

8.2.2 Commands

When you press (right)CTRL+C, you can type the following command:

Command	Explanation
auto	**AUTO** make calculation automatically
blkmap y;x	**BLKMAP** copy (cy;cx)-(y;x) cells (11 chars) to Word Processor
ca	**CA**talogue list of the stored tables
cco x	**C**olumn **CO**py copy x column to the current column
cdel	**C**olumn **DEL**ete delete the current column
cins	**C**olumn **INS**ert insert a new column at the current column
cm	**C**lear **M**emory clear the table from the memory
color n	**C**olour change the colours (n=0..15)
copy y;x	**COPY** copy (y;x) cell to the active cell
df	**D**elete **F**ile delete a table
fit y;x	**FIT**ing a cell copy and convert a formula
fu	**FU**ll screen switch to full screen
goto y;x	**GOTO** go to y line and x column
gr	**GR**aph make a column-chart from the current row
grc	**GR**aph **C**olumn-chart same as GR, but this is a nicer one
ha	**HA**lf screen switch to half screen (multitasking)
home	**HOME** go to home (1;1)

Command	Explanation
leftj	**LEFT J**ustify align a cell to the left
lf	**L**oad **F**ile load a table
man	**MAN** make calculation manually
map y;x	**MAP** copy (cy;cx)-(y;x) cells (36 chars) to Word Processor
poke o,v	**POKE** set a value of [0] segment
rco x	**R**ow **CO**py copy y row to the current row
rdel	**R**ow **DEL**ete delete the current row
rightj	**RIGHT J**ustify align a cell to the right
rins	**R**ow **INS**ert insert a new row at the current row
sf	**S**ave **F**ile save the table
tf	**T**o our **F**ilemanager switch to Database
tw	**T**o our **W**ord processor switch to Word Processor

8.2.3 Functions

Function	Code	Explanation
+	0x80	Addition
-	0x81	Subtraction
*	0x82	Multiplying
/	0x83	Division
↑	0x84	Power
abs y;x	0x85	**ABS**olute value
atn y;x	0x86	**A**rc **T**a**N**gent
cos y;x	0x87	**COS**inus
div y1;x1 to y2;x2	0x88	**DIV**ision of cells
exp y;x	0x89	**EXP**onent of e (2.71828)
log y;x	0x8A	**LOG**arithm
max y1;x1 to y2;x2	0x8B	**MAX**imum of cells
min y1;x1 to y2;x2	0x8C	**MIN**imum of cells
mlt y1;x1 to y2;x2	0x8D	**MuLT**iply of cells
sin y;x	0x8E	**SIN**inus
sub y1;x1 to y2;x2	0x8F	**SUB**traction of cells
sum y1;x1 to y2;x2	0x90	**SUM**marize of cells
tan y;x	0x91	**TAN**gent
iftrue	0x92	**IF** the condition is **TRUE**
notiftrue	0x93	**NOT IF** the condition is **TRUE**

+ - * / ↑ - To make calculations:

<y1>;<x1> <+ - * / ↑> <y2>;<x2> [#n]

y	Y row
x	X column
+	addition
-	subtraction
*****	multiplying
/	division
↑	power
#n	a constant number

eg.:

6;3+17;11 : (6;3)+(17;11)

23;25-150;1 : (23;25)-(150;1)

234;18*20;19 : (234;18)*(20;19)

3;1/4;1 : (3;1)/(4;1)

567;10↑879;20 : (567;10)↑(879;20)

1;2+3;4-5;6*7;8/9;10↑11;12 : (1;2)+(3;4)-(5;6)*(7;8)/(9;10)↑(11;12)

10;20↑#2 : (10;20)↑2

ABS - To make absolute value:

ABS <y>;<x> [#n]

y	Y row
x	X column
#n	a constant number

eg.:
ABS 2;3 : absolute value of the (2;3) cell
ABS #-43 : absolute value of -43 (43)

ATN - To make arc tangent:

ATN <y>;<x> [#n]

y	Y row
x	X column
#n	a constant number

eg.:
ATN 8;4 : arc tangent of the (8;4) cell
ATN #45 : arc tangent of 45 (1)

COS - To make cosinus:

COS <y>;<x> [#n]

y	Y row
x	X column
#n	a constant number

eg.:
COS 21;25 : cosinus of the (21;25) cell
COS #90 : cosinus of 90 (0)

DIV - To make division of cells:

DIV <y1>;<x1> TO <y2>;<x2>

y1 Y1 row
x1 X1 column
y2 Y2 row
x2 X2 column

eg.:
DIV 2;3 TO 2;5 : ((2;3) / (2;4)) / (2;5)
DIV 3;2 TO 5;2 : ((3;2) / (4;2)) / (5;2)

EXP - To make exponent of e (2.71828) :

EXP <y>;<x> [#n]

y Y row
x X column
#n a constant number

eg.:
EXP 4;2 : exponent of e by the (4;2) cell
EXP #4 : exponent of e by 4 (54.5980)

LOG - To make logarithm:

LOG <y>;<x> [#n]

y Y row
x X column
#n a constant number

eg.:
LOG 1;7 : logarithm of the (1;7) cell
LOG #1000 : logarithm of 1000 (3)

MAX - To take the maximum value of cells:

MAX <y1>;<x1> TO <y2>;<x2>

y1 Y1 row (left-top)
x1 X1 column (left-top)
y2 Y2 row (right-bottom)
x2 X2 column (right-bottom)

eg.:
MAX 2;1 TO 2;25 : get the maximum value of the 2nd row
MAX 1;4 TO 1000;4 : get the maximum value of the 4th column
MAX 2;2 TO 5;5 : get the maximum value of 16 cells

MIN - To take the minimum value of cells:

MIN <y1>;<x1> TO <y2>;<x2>

y1 Y1 row (left-top)
x1 X1 column (left-top)
y2 Y2 row (right-bottom)
x2 X2 column (right-bottom)

eg.:
MIN 21;1 TO 21;25 : get the minimum value of the 21st row
MIN 1;13 TO 1000;13 : get the minimum value of the 13th column
MIN 2;2 TO 6;6 : get the minimum value of 25 cells

MLT - To make multiplying of cells:

MLT <y1>;<x1> TO <y2>;<x2>

y1 Y1 row (top-left)
x1 X1 column (top-left)
y2 Y2 row (right-bottom)
x2 X2 column (right-bottom)

eg.:
MLT 2;3 TO 2;5 : (2;3)*(2;4)*(2;5)
MLT 3;2 TO 5;2 : (3;2)*(4;2)*(5;2)
MLT 3;3 TO 4;4 : (3;3)*(3;4)*(4;3)*(4;4)

SIN - To make sinus:

SIN <y>;<x> [#n]

y	Y row
x	X column
#n	a constant number

eg.:
SIN 25;21 : sinus of the (25;21) cell
SIN #90 : sinus of 90 (1)

SUB - To make subtraction of cells:

SUB <y1>;<x1> TO <y2>;<x2>

y1	Y1 row
x1	X1 column
y2	Y2 row
x2	X2 column

eg.:
SUB 2;3 TO 2;5 : ((2;3) - (2;4)) - (2;5)
SUB 3;2 TO 5;2 : ((3;2) - (4;2)) - (5;2)

SUM - To make sum of cells:

SUM <y1>;<x1> TO <y2>;<x2>

y1	Y1 row (left-top)
x1	X1 column (left-top)
y2	Y2 row (right-bottom)
x2	X2 column (right-bottom)

eg.:
SUM 2;3 TO 2;5 : (2;3)+(2;4)+(2;5)
SUM 3;2 TO 5;2 : (3;2)+(4;2)+(5;2)
SUM 3;3 TO 4;4 : (3;3)+(3;4)+(4;3)+(4;4)

TAN - To make tangent:

TAN <y>;<x> [#n]

y	Y row
x	X column
#n	a constant number

eg.:

TAN 4;8 : tangent of the (4;8) cell

TAN #45 : arc tangent of 45 (1)

[NOT]IFTRUE - To make condition:

<y1>;<x1> <cond> [#n] [NOT]IFTRUE <y2>;<x2> ← <y3>;<x3>

y	Y row of a cell
x	X column of a cell
cond	condition: = > < nte not
#n	a constant number

eg.:

3;4 < #50 IFTRUE 10;11 ← #1

12;5 nte #16 NOTIFTRUE 25;1 ← 4;10

127;23+56;2 = COS #45 IFTRUE 7;4 ← SIN 3;2

8.3 Database

Database enables you to store a large number of datas. You can store up to 10 000 records. One record has got 25 fields. One field can be maximum 38 character long.

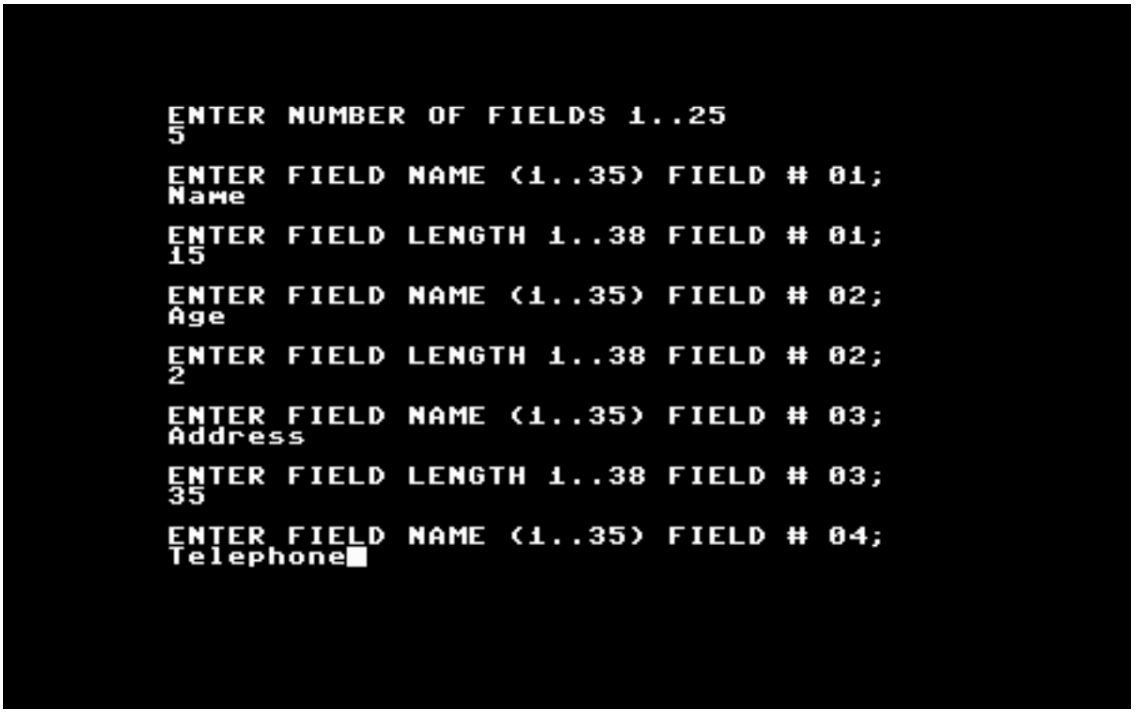

Figure 8.6: Database - NEWTF

The records stored in a database are able to use in the Word Processor, for example when you make a circular letter, such as an invitation for a party.

You can modify a record, search a record, review records or sort records.

8.3.1 Keys

Key	Explanation
(right) CTRL+C	Enter to command line
(right) CTRL+Q	Repeating the last command
(right) CTRL+S	Stop ReView records (RV command)

8.3.2 Commands

When you press (right)CTRL+C, you can type the following command:

Command	Explanation
ca	**CA**talogue list of the saved databases
df	**D**elete **F**ile delete a database
ds f1[;f2[;f3]]	**D**isk **S**ort sort records by field1 [and field2 [and field3]]
highrc n	**HIGH R**e**C**ord set the highest number of records
lf	**L**oad **F**ile load a database
newtf	**NEW T**o our **F**ilemanager create a new database
nr	**N**ext **R**ecord save the current record and jump to the next one
pi fn	**PI**ck set TOP and BOTTOM range of a field
poke o,v	**POKE** set a value of [0] segment
resetlist	**RESET LIST** set to default (no PI, no HIGHRC)
rc [n]	**R**e**C**ord edit the current record (or the n-th)
rv [n]	**R**e**V**iew review records from the current (or n-th)
sf	**S**ave **F**ile save the database
sr	**S**ea**R**ch search a text in records
tc	**T**o our **C**omputational switch to Spreadsheet
tf	**T**o our **F**ilemanager load a database or give information about it
tw	**T**o our **W**ord processor switch to Word Processor

Command	**Explanation**
ud [n]	**U**p**D**ate record save the current record (to n-th position)

8.4 Graph

Graph enables you to represent values on a chart. You can represent values of the current row. For example: 1,2,3,4,5,6,7,8,9,10,10,8,6,4,2,0,8,15

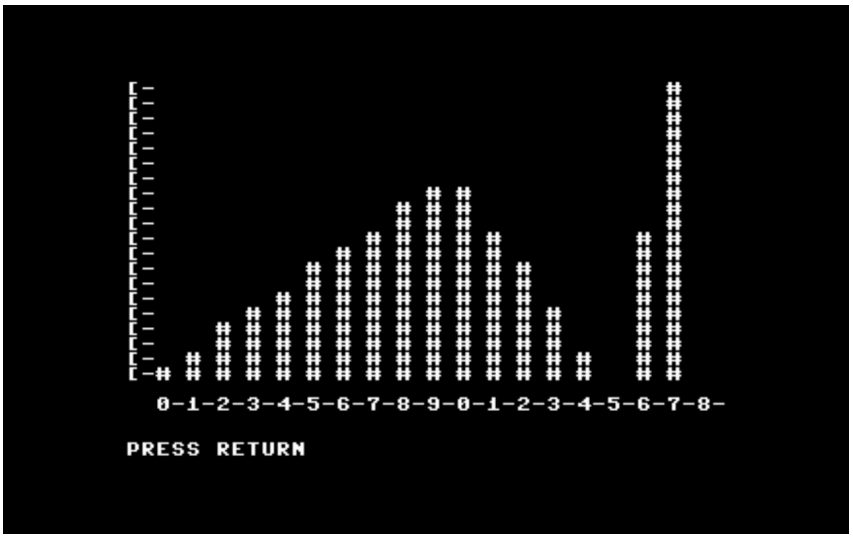

Figure 8.7: Graph (GR)

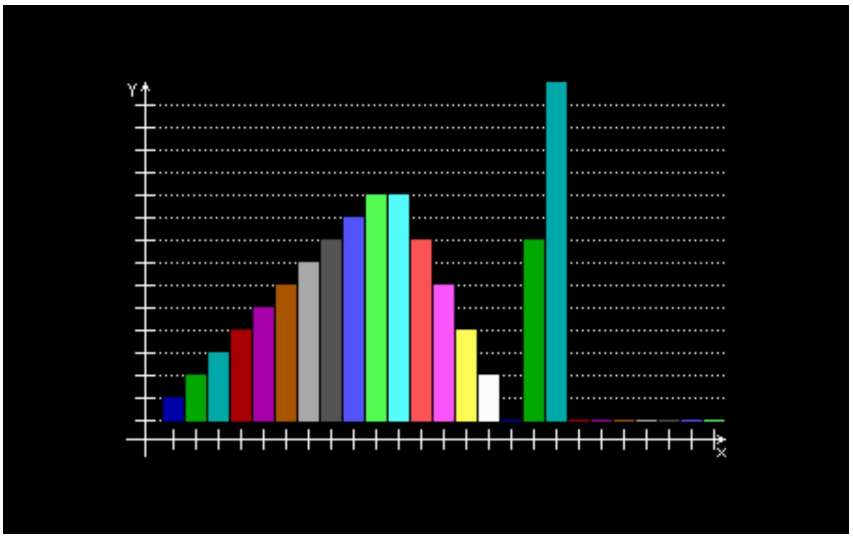

Figure 8.8: Graph (GRC)

When you press Enter, you will be back to Spreadsheet. After GR command, the chart appears in the Word Processor at the current position of the cursor with all (25) columns!

8.5 Special effects

In this section I would like to draw attention to some interesting things. Such as a new chart (GRC) mentioned the pervious section, multitasking when you can see the Word Processor and the Spreadsheet together, and so on...

8.5.1 Multitasking

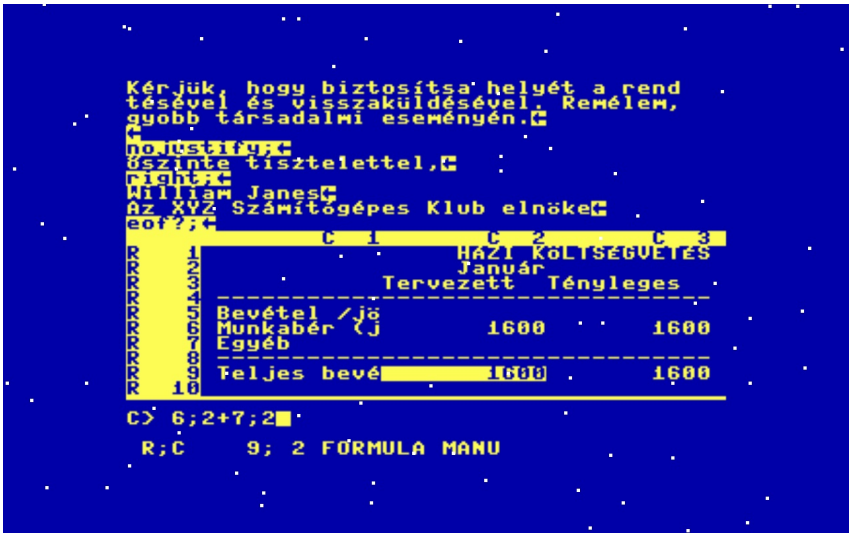

Figure 8.9: Multitasking (Spreadsheet - HA) + Snowfalling (POKE)

When you type FU command, you will switch back to single task-mode.

8.5.2 Intelligent RLE packer for 3-PLUS-1

I made an intelligent simple packer for Word Processor, Spreadsheet and Database. It needed because the saved files were too large. This variation of RLE packer is intelligent because it listens to the amount of the redundant datas in the files and it doesn't do anything in that case there are not any same datas.

- Word Processor file was 131 072 bytes, and now can be only 929 bytes.
- Spreadsheet file was 1 966 080 bytes, and now can be only 97 bytes.
- Database file was 9 568 256 bytes, and now can be only 265 bytes.

Naturally the extract ratio is not always so good, it is depend on how different the datas are in the files.

Chapter 9

Appendix

In the last chapter you can see the tables of instructions and other supplementary materials. You will realize that TV4B is able to be improved. New instructions is typed in bold. There are many leaks in Assembly instructions and memory map waiting for new capabilities.

9.1 BASIC instructions

name	ID
END	80h
FOR	81h
NEXT	82h
DATA	83h
INPUT#	84h
INPUT	85h
DIM	86h
READ	87h
LET	88h
GOTO	89h
RUN	8Ah
IF	8Bh
RESTORE	8Ch
GOSUB	8Dh
RETURN	8Eh
REM	8Fh
STOP	90h
ON	91h
WAIT	92h
LOAD	93h
SAVE	94h
VERIFY	95h
DEF	96h
POKE	97h
PRINT#	98h
PRINT	99h
CONT	9Ah
LIST	9Bh
CLR	9Ch
CMD	9Dh
SYS	9Eh
OPEN	9Fh

name	**ID**
CLOSE	A0h
GET	A1h
NEW	A2h
TAB(	A3h
TO	A4h
FN	A5h
SPC(	A6h
THEN	A7h
NOT	A8h
STEP	A9h
+	AAh
-	ABh
*	ACh
/	ADh
(power)	AEh
AND	AFh
OR	B0h
>	B1h
=	B2h
<	B3h
SGN	B4h
INT	B5h
ABS	B6h
USR	B7h
FRE	B8h
POS	B9h
SQR	BAh
RND	BBh
LOG	BCh
EXP	BDh
COS	BEh
SIN	BFh

name	**ID**
TAN	C0h
ATN	C1h
PEEK	C2h
LEN	C3h
STR$	C4h
VAL	C5h
ASC	C6h
CHR$	C7h
LEFT$	C8h
RIGHT$	C9h
MID$	CAh
SCREEN (GO)	CBh
RGR	CCh
RCLR	CDh
SOCKET (RLUM)	CEh
JOY	CFh
RDOT	D0h
DEC	D1h
HEX$	D2h
ERR$	D3h
INSTR	D4h
ELSE	D5h
RESUME	D6h
TRAP	D7h
TRON	D8h
TROFF	D9h
SOUND	DAh
VOL	DBh
AUTO	DCh
PUDEF	DDh
GRAPHIC	DEh
PAINT	DFh

name	**ID**
CHAR	E0h
BOX	E1h
CIRCLE	E2h
GSHAPE	E3h
SSHAPE	E4h
DRAW	E5h
LOCATE	E6h
COLOR	E7h
SCNCLR	E8h
IMAGE (SCALE)	E9h
HELP	EAh
DO	EBh
LOOP	ECh
EXIT	EDh
DIRECTORY	EEh
DSAVE	EFh
DLOAD	F0h
MKDIR (HEADER)	F1h
SCRATCH	F2h
CHDIR (COLLECT)	F3h
COPY	F4h
RENAME	F5h
MUSIC (BACKUP)	F6h
DELETE	F7h
RENUMBER	F8h
KEY	F9h
MONITOR	FAh
USING	FBh
UNTIL	FCh
WHILE	FDh
SEGMENT	FEh
??	FFh

9.2 ASSEMBLY instructions

name	ID	size	remark
BRK	00h	1	force an interrupt or BReaK (the program)
ORA ($xx,X)	01h	2	”OR” memory with Accumulator
???	02h	?	?
???	03h	?	?
???	04h	?	?
ORA $xx	05h	2	”OR” memory with Accumulator
ASL $xx	06h	2	Shift one bit Left (memory or Accumulator)
???	07h	?	?
PHP	08h	1	PusH Processor status on stack
ORA #$xx	09h	2	”OR” memory with Accumulator
ASL	0Ah	1	Shift one bit Left (memory or Accumulator)
???	0Bh	?	?
???	0Ch	?	?
ORA $xxxx	0Dh	3	”OR” memory with Accumulator
ASL $xxxx	0Eh	3	Shift one bit Left (memory or Accumulator)
???	0Fh	?	?
BPL $xxxx	10h	3	Branch on result PLus
ORA ($xx),Y	11h	2	”OR” memory with Accumulator
???	12h	?	?
???	13h	?	?
???	14h	?	?
ORA $xx,X	15h	2	”OR” memory with Accumulator
ASL $xx,X	16h	2	Shift one bit Left (memory or Accumulator)
???	17h	?	?
CLC	18h	1	CLear Carry flag
ORA $xxxx,Y	19h	3	”OR” memory with Accumulator
???	1Ah	?	?
???	1Bh	?	?
???	1Ch	?	?
ORA $xxxx,X	1Dh	3	”OR” memory with Accumulator
ASL $xxxx,X	1Eh	3	Shift one bit Left (memory or Accumulator)
???	1Fh	?	?

name	**ID**	**size**	**remark**
JSR $xxxx	20h	3	Jump to a SubRoutine
AND ($xx,X)	21h	2	"AND" memory with Accumulator
???	22h	?	?
???	23h	?	?
BIT $xx	24h	2	test BITs in memory with Accumulator
AND $xx	25h	2	"AND" memory with Accumulator
ROL $xx	26h	2	ROtate one bit Left (memory or Accumulator)
???	27h	?	?
PLP	28h	1	PuLl Processor status from stack
AND #$xx	29h	2	"AND" memory with Accumulator
ROL	2Ah	1	ROtate one bit Left (memory or Accumulator)
???	2Bh	?	?
BIT $xxxx	2Ch	3	test BITs in memory with Accumulator
AND $xxxx	2Dh	3	"AND" memory with Accumulator
ROL $xxxx	2Eh	3	ROtate one bit Left (memory or Accumulator)
???	2Fh	?	?
BMI $xxxx	30h	3	Branch on result MInus
AND ($xx),Y	31h	2	"AND" memory with Accumulator
???	32h	?	?
???	33h	?	?
???	34h	?	?
AND $xx,X	35h	2	"AND" memory with Accumulator
ROL $xx,X	36h	2	ROtate one bit Left (memory or Accumulator)
???	37h	?	?
SEC	38h	1	SEt Carry flag
AND $xxxx,Y	39h	2	"AND" memory with Accumulator
???	3Ah	?	?
???	3Bh	?	?
???	3Ch	?	?
AND $xxxx,X	3Dh	3	"AND" memory with Accumulator
ROL $xxxx,X	3Eh	3	ROtate one bit Left (memory or Accumulator)
???	3Fh	?	?

name	ID	size	remark
RTI	40h	1	ReTurn from Interrupt
EOR ($xx,X)	41h	2	"Exclusive-OR" memory with Accumulator
???	42h	?	?
???	43h	?	?
???	44h	?	?
EOR $xx	45h	2	"Exclusive-OR" memory with Accumulator
LSR $xx	46h	2	Shift one bit Right (memory or Accumulator)
???	47h	?	?
PHA	48h	1	PusH Accumulator on stack
EOR #$xx	49h	2	"Exclusive-OR" memory with Accumulator
LSR	4Ah	1	Shift one bit Right (memory or Accumulator)
???	4Bh	?	?
JMP $xxxx	4Ch	3	JuMP to new location
EOR $xxxx	4Dh	3	"Exclusive-OR" memory with Accumulator
LSR $xxxx	4Eh	3	Shift one bit Right (memory or Accumulator)
???	4Fh	?	?
BVC $xxxx	50h	3	Branch on oVerflow Clear
EOR ($xx),Y	51h	2	"Exclusive-OR" memory with Accumulator
???	52h	?	?
???	53h	?	?
???	54h	?	?
EOR $xx,X	55h	2	"Exclusive-OR" memory with Accumulator
LSR $xx,X	56h	2	Shift one bit Right (memory or Accumulator)
???	57h	?	?
CLI	58h	1	CLear Interrupt disable bit
EOR $xxxx,Y	59h	3	"Exclusive-OR" memory with Accumulator
???	5Ah	?	?
???	5Bh	?	?
???	5Ch	?	?
EOR $xxxx,X	5Dh	3	"Exclusive-OR" memory with Accumulator
LSR $xxxx,X	5Eh	3	Shift one bit Right (memory or Accumulator)
???	5Fh	?	?

name	ID	size	remark
RTS	60h	1	ReTurn from Subroutine
ADC ($xx,X)	61h	2	ADd with Carry to Accumulator
???	62h	?	?
???	63h	?	?
???	64h	?	?
ADC $xx	65h	2	ADd with Carry to Accumulator
ROR $xx	66h	2	ROtate one bit Right (memory or Accumulator)
???	67h	?	?
PLA	68h	1	PuLl Accumulator from stack
ADC #$xx	69h	2	ADd with Carry to Accumulator
ROR	6Ah	1	ROtate one bit Right (memory or Accumulator)
???	6Bh	?	?
JMP ($xxxx)	6Ch	3	JuMP to new location
ADC $xxxx	6Dh	3	ADd with Carry to Accumulator
ROR $xxxx	6Eh	3	ROtate one bit Right (memory or Accumulator)
???	6Fh	?	?
BVS $xxxx	70h	3	Branch on oVerflow Set
ADC ($xx),Y	71h	2	ADd with Carry to Accumulator
???	72h	?	?
???	73h	?	?
???	74h	?	?
ADC $xx,X	75h	2	ADd with Carry to Accumulator
ROR $xx,X	76h	2	ROtate one bit Right (memory or Accumulator)
???	77h	?	?
SEI	78h	1	SEt Interrupt disable status
ADC $xxxx,Y	79h	3	ADd with Carry to Accumulator
???	7Ah	?	?
???	7Bh	?	?
???	7Ch	?	?
ADC $xxxx,X	7Dh	3	ADd with Carry to Accumulator
ROR $xxxx,X	7Eh	3	ROtate one bit Right (memory or Accumulator)
???	7Fh	?	?

name	ID	size	remark
???	80h	?	?
STA ($xx,X)	81h	2	STore Accumulator in memory
???	82h	?	?
???	83h	?	?
STY $xx	84h	2	STore index Y in memory
STA $xx	85h	2	STore Accumulator in memory
STX $xx	86h	2	STore index X in memory
???	87h	?	?
DEY	88h	1	DEcrement index Y by one
???	89h	?	?
TXA	8Ah	1	Transfer index X to Accumulator
???	8Bh	?	?
STY $xxxx	8Ch	3	STore index Y in memory
STA $xxxx	8Dh	3	STore Accumulator in memory
STX $xxxx	8Eh	3	STore index X in memory
???	8Fh	?	?
BCC $xxxx	90h	3	Branch on Carry Clear
STA ($xx),Y	91h	2	STore Accumulator in memory
???	92h	?	?
???	93h	?	?
STY $xx,X	94h	2	STore index Y in memory
STA $xx,X	95h	2	STore Accumulator in memory
STX $xx,Y	96h	2	STore index X in memory
???	97h	?	?
TYA	98h	1	Transfer index Y to Accumulator
STA $xxxx,Y	99h	3	STore Accumulator in memory
TXS	9Ah	1	Transfer index X to Stack Pointer
???	9Bh	?	?
???	9Ch	?	?
STA $xxxx,X	9Dh	3	STore Accumulator in memory
???	9Eh	?	?
???	9Fh	?	?

name	ID	size	remark
LDY #$xx	A0h	2	LoaD memory to index Y
LDA ($xx,X)	A1h	2	LoaD memory to Accumulator
LDX #$xx	A2h	2	LoaD memory to index X
???	A3h	?	?
LDY $xx	A4h	2	LoaD memory to index Y
LDA $xx	A5h	2	LoaD memory to Accumulator
LDX $xx	A6h	2	LoaD memory to index X
???	A7h	?	?
TAY	A8h	1	Transfer Accumulator to index Y
LDA #$xx	A9h	2	LoaD memory to Accumulator
TAX	AAh	1	Transfer Accumulator to index X
???	ABh	?	?
LDY $xxxx	ACh	3	LoaD memory to index Y
LDA $xxxx	ADh	3	LoaD memory to Accumulator
LDX $xxxx	AEh	3	LoaD memory to index X
???	AFh	?	?
BCS $xxxx	B0h	3	Branch on Carry Set
LDA ($xx),Y	B1h	2	LoaD memory to Accumulator
???	B2h	?	?
???	B3h	?	?
LDY $xx,X	B4h	2	LoaD memory to index Y
LDA $xx,X	B5h	2	LoaD memory to Accumulator
LDX $xx,Y	B6h	2	LoaD memory to index X
???	B7h	?	?
CLV	B8h	1	CLear oVerflow flag
LDA $xxxx,Y	B9h	3	LoaD memory to Accumulator
TSX	BAh	1	Transfer Stack Pointer to index X
???	BBh	?	?
LDY $xxxx,X	BCh	3	LoaD memory to index Y
LDA $xxxx,X	BDh	3	LoaD memory to Accumulator
LDX $xxxx,Y	BEh	3	LoaD memory to index X
???	BFh	?	?

name	ID	size	remark
CPY #$xx	C0h	2	ComPare memory and index Y
CMP ($xx,X)	C1h	2	ComPare memory and Accumulator
???	C2h	?	?
???	C3h	?	?
CPY $xx	C4h	2	ComPare memory and index Y
CMP $xx	C5h	2	ComPare memory and Accumulator
DEC $xx	C6h	2	DECrement memory by one
???	C7h	?	?
INY	C8h	1	INcrement index Y by one
CMP #$xx	C9h	2	ComPare memory and Accumulator
DEX	CAh	1	DEcrement index X by one
???	CBh	?	?
CPY $xxxx	CCh	3	ComPare memory and index Y
CMP $xxxx	CDh	3	ComPare memory and Accumulator
DEC $xxxx	CEh	3	DECrement memory by one
???	CFh	?	?
BNE $xxxx	D0h	3	Branch on result not zero
CMP ($xx),Y	D1h	2	ComPare memory and Accumulator
???	D2h	?	?
???	D3h	?	?
???	D4h	?	?
CMP $xx,X	D5h	2	ComPare memory and Accumulator
DEC $xx,X	D6h	2	DECrement memory by one
???	D7h	?	?
CLD	D8h	1	CLear Decimal mode
CMP $xxxx,Y	D9h	3	ComPare memory and Accumulator
???	DAh	?	?
???	DBh	?	?
???	DCh	?	?
CMP $xxxx,X	DDh	3	ComPare memory and Accumulator
DEC $xxxx,X	DEh	3	DECrement memory by one
???	DFh	?	?

name	**ID**	**size**	**remark**
CPX #$xx	E0h	2	ComPare memory and index X
SBC ($xx,X)	E1h	2	SuBtract memory and Carry from Accumulator
???	E2h	?	?
???	E3h	?	?
CPX $xx	E4h	2	ComPare memory and index X
SBC $xx	E5h	2	SuBtract memory and Carry from Accumulator
INC $xx	E6h	2	INCrement memory by one
???	E7h	?	?
INX	E8h	1	INcrement index X by one
SBC #$xx	E9h	2	SuBtract memory and Carry from Accumulator
NOP	EAh	1	No OPeration
???	EBh	?	?
CPX $xxxx	ECh	3	ComPare memory and index X
SBC $xxxx	EDh	3	SuBtract memory and Carry from Accumulator
INC $xxxx	EEh	3	INCrement memory by one
???	EFh	?	?
BEQ $xxxx	F0h	3	Branch on result zero
SBC ($xx),Y	F1h	2	SuBtract memory and Carry from Accumulator
???	F2h	?	?
???	F3h	?	?
???	F4h	?	?
SBC $xx,X	F5h	2	SuBtract memory and Carry from Accumulator
INC $xx.X	F6h	2	INCrement memory by one
???	F7h	?	?
SED	F8h	1	SEt Decimal mode
SBC $xxxx,Y	F9h	3	SuBtract memory and Carry from Accumulator
???	FAh	?	?
???	FBh	?	?
???	FCh	?	?
SBC $xxxx,X	FDh	3	SuBtract memory and Carry from Accumulator
INC $xxxx.X	FEh	3	INCrement memory by one
???	FFh	?	?

9.3 Full memory map

```
[000]:System Area
      0000 --> Embedded programs (Word processor,Spreadsheet,Database,Graph)
                 0 --> Loading embedded programs
                 1 --> Word processor
                 2 --> Spreadsheet
                 3 --> Database
                 4 --> Graph
                96 --> Default RTS (0x60)
      0001 --> Reading/Writing a file from disk (look at 0x025D)
      0002 --> UNTIL/WHILE flag for exit (0:normal; 1:exit from UNTIL/WHILE)
      0003 --> Increment for RENUMBER (low)
      0004 --> Increment for RENUMBER (high)
      0005 --> Increment for RENUMBER (seg)
       .
       .
       .
      0008 --> 0: no TAB before; >0: TAB before
      0009 --> TAB column counter
      000A --> (D)LOAD/(D)SAVE flag (0:LOAD/SAVE; 1:DLOAD/DSAVE)
       .
       .
       .
      000C --> Array flag (0: out of array; >0: in an array)
       .
       .
       .
      0010 --> Subroutine counter
       .
       .
       .
      002A --> Segment of BASIC start
      002B --> Low byte of BASIC start address
      002C --> High byte of BASIC start address
       .
       .
       .
      0037 --> Current BASIC line number
      0038 --> Current BASIC line number
      0039 --> Current BASIC line number
      003A --> STOP/CONT offset in the current BASIC line (low)
      003B --> STOP/CONT offset in the current BASIC line (high)
       .
       .
       .
      003F --> Current DATA segment
      0040 --> Current DATA offset (low byte)
      0041 --> Current DATA offset (high byte)
      0042 --> Current DATA line length (low byte)
      0043 --> Current DATA line length (high byte)
      0044 --> INPUT flag
                 0=direct mode;
                 1=input;
                 2=sound;
```

```
           3=run/go;
           4=embedded program (no cursor)
           5=embedded program (no cursor and RETURN)
         255=EOP)
 .
 .
 .
0049 --> Current iteration variable type
004A --> Current iteration variable pointer for FOR/NEXT (4.byte)
004B --> Current iteration variable pointer for FOR/NEXT (3.byte)
004C --> Current iteration variable pointer for FOR/NEXT (2.byte)
004D --> Current iteration variable pointer for FOR/NEXT (1.byte)
004E --> Iteration variable exists or not (>0: not exists before)
004F --> IF value for THEN (0:false; 1:true)
0050 --> Original length of the current compiled code (low byte)
0051 --> Original length of the current compiled code (high byte)
0052 --> DEBUG flag (0:TROFF; >0:TRON;)
0053 --> EXIT flag (0:normal; >0:exit;)
0054 --> Current position of the current original compiled code (low byte)
0055 --> Current position of the current original compiled code (high byte)
0056 --> Current position of the current compiled code (low byte)
0057 --> Current position of the current compiled code (high byte)
 .
 .
 .
005C --> Flag for logical expressions in arithmetic
 .
 .
 .
0060 --> USR flag (0:nothing; 1:calling)
0061 --> USR parameter
0062 --> USR parameter
0063 --> USR parameter
0064 --> USR parameter
 .
 .
 .
0073 --> Low byte of increment value for AUTO
0074 --> High byte of increment value for AUTO
 .
 .
 .
0078 --> Sound channel
         0 - Mono
         1 - Left
         2 - Right
         3 - Stereo#1 (1 sound channel to left and 2+3 to right)
         4 - Stereo#2 (2+3 sound channels to left and 1 to right)
0079 --> Flag for Sound Timer #1
007A --> Flag for Sound Timer #2
007B --> Flag for Sound Playing (0: no playing; >0: playing;)
007C --> Low byte of time counter (eg. sound delay)
007D --> High byte of time counter
007E --> Volume (*32)
007F --> Flag for running sound from BASIC (0:not; >0:sound;)
0080 --> Music type (0:nothing; 1:WAV; 2:MOD; 3:MP3;)
```

```
0081 --> Run on startup (0:nothing; 1:BN; 2:BR; 3:AN; 4:AR; 5:DN; 6:DR;)
0082 --> Return flag (0: nothing; >0: pressing enter)
0083 --> Screen mode (if graphics on)
         0      - full graphics no text
         1..12  - text row numbers from bottom of the screen
         13..24 - text row numbers (-12) from top of the screen
         25..44 - text column number (-24) from left of the screen
         45..64 - text column number (-44) from right of the screen
         65..74 - text row, column number (-64) from top, bottom,
                  left and right
         75..   - text screen in the graphics screen given 07E5..07E8
0084 --> TrueColor mode
         0  - not set
         >0 - set (activated when graphic on)
0085 --> Foreground color for text
0086 --> Background color for text
 .
 .
 .
0090 --> Wait for main thread (INPUT, GETKEY, WAIT)
 .
 .
 .
0092 --> RTS flag (from BASIC or Assembly)
 .
 .
 .
0096 --> Last BASIC instruction
0097 --> Previous BASIC instruction
 .
 .
 .
0099 --> CMD flag (0=screen; 1=screen+printer; 2=screen+file)
009A --> 0x00 = program mode; 0x80 = direct mode
 .
 .
 .
00A3 --> Timer (65536)
00A4 --> Timer (256)
00A5 --> Timer (1) --> 1/100 (10ms)
 .
 .
 .
00A9 --> Frames Per Second (FPS)
00AA --> Sound Frag Number (SFN)
 .
 .
 .
00B1 --> Error number for ERR$
           0: NO ERROR
           4: ?FILE NOT FOUND --> File is not found
          10: ?NEXT WITHOUT FOR --> NEXT without FOR
          11: ?SYNTAX --> illegal commands or instructions
          12: ?RETURN WITHOUT GOSUB --> RETURN without GOSUB
          20: ?DIVISION BY ZERO --> Division by zero
          26: ?CAN'T CONTINUE --> CONT without running
```

```
           28: ?VERIFY --> the program is different from the memory
           29: ?LOAD --> something wrong during loading
           30: ?BREAK --> the user interrupts the program by Escape
           31: ?CAN'T RESUME --> RESUME without TRAP
           35: ?NO GRAPHIC AREA --> DRAW, BOX, etc. without GRAPHIC
 .
 .
 .
00C6 --> Keyboard ASCII code
00C7 --> Special keyboard code (0: normal; >0: special codes)
 .
 .
 .
00CA --> Current cursor X position on text screen
 .
 .
 .
00CD --> Current cursor Y position on text screen
 .
 .
 .
00CF --> Flag for interrupt
 .
 .
 .
00EE --> Flag for keyboard IRQ (0: disabled; 1: enabled)
00EF --> Keyboard index (0..9)
 .
 .
 .
00FF --> Flag for control characters (0:off; >0:on)
0100 --> System stack
 .       (256bytes)
 .
 .
01FF --> End of system stack
 .
 .
 .
025D --> File#1 EXIST flag (0: not open file; 1: open file for reading)
025E --> File#1 EOF flag (0: not EOF; 1: End Of File)
025F --> File#1 SEEK flag (0: reading in sequentially; 1: seek set)
0260 --> File#1 position for SEEK set
 .       (DWORD)
 .
 .
0264 --> File#2 EXIST flag (0: not open file; 1: open file for writing)
0265 --> File#2 SEEK flag (0: writing in sequentially; 1: seek set)
0266 --> File#2 position for SEEK set
 .       (DWORD)
 .
 .
026A --> File handles for OPEN/CLOSE (0..255)
026B --> Temp area for kernel storing
 .       (64bytes)
 .
```

```
 .
02AB --> FOR/DRAW flag (TO) --> 0: FOR; >0: DRAW
02AC --> Color number for TO (0:BackGround, 1:Cursor, 4:Frame)
02AD --> Current X position of graphic (low byte)
02AE --> Current X position of graphic (high byte)
02AF --> Current Y position of graphic (low byte)
02B0 --> Current Y position of graphic (high byte)
 .
 .
 .
02CC --> X coordinate of circle center (low byte)
02CD --> X coordinate of circle center (high byte)
02CE --> Y coordinate of circle center (low byte)
02CF --> Y coordinate of circle center (high byte)
02D0 --> X width of radius (low byte)
02D1 --> X width of radius (high byte)
02D2 --> Y width of radius (low byte)
02D3 --> Y width of radius (high byte)
02D4 --> Rotation angle (low byte)
02D5 --> Rotation angle (high byte)
02D6 --> Angle of polygon (low byte)
02D7 --> Angle of polygon (high byte)
02D8 --> Arc angle start (low byte)
02D9 --> Arc angle start (high byte)
02DA --> Arc angle end (low byte)
02DB --> Arc angle end (high byte)
 .
 .
 .
030C --> ESCAPE flag (0:normal; >0: escape)
 .
 .
 .
0312 --> Data segment for interrupt (00)
0313 --> Code segment for interrupt (00)
0314 --> Low byte of the interrupt address (0E)
0315 --> High byte of the interrupt address (CE)
 .
 .
 .
032C --> X coordinate of the character for printer
032D --> Y coordinate of the character for printer
 .
 .
 .
04DC --> Sound Sample Rate (hb of hw)
04DD --> Sound Sample Rate (lb of hw)
04DE --> Sound Sample Rate (hb of lw)
04DF --> Sound Sample Rate (lb of lw)
04E0 --> Sound Channel (low)
04E1 --> Sound Channel (high)
04E2 --> Sound Bits (low)
04E3 --> Sound Bits (high)
04E4 --> Sound Properties flag (0:none; >0:use)
 .
 .
```

```
 .
04E7 --> [SPACE] symbol used by USING
04E8 --> [,] symbol used by USING
04E9 --> [.] symbol used by USING
04EA --> [$] symbol used by USING
 .
 .
 .
04EE --> Error number flag for 00B1 (0: OK; >0: Stop the program)
04EF --> BASIC line number of last error (low: PC low)
04F0 --> BASIC line number of last error (middle: PC high)
04F1 --> BASIC line number (high: segment)
04F2 --> TRAP BASIC line number (low: PC low)
04F3 --> TRAP BASIC line number (middle: PC high)
04F4 --> TRAP BASIC line number (high: segment)
04F5 --> offset in BASIC line of last error (low)
04F6 --> offset in BASIC line of last error (high)
04F7 --> TRAP flag, STOP flag
         0.bit: 0: no trap; 1: trap;
         1.bit: 0: not call the trap; 1: call the trap;
         4.bit: 0: no stop; 1: stop;
 .
 .
04FA --> The first counter (FF00) restart (low byte)
04FB --> The first counter (FF01) restart (high byte)
04FC --> Sound Timer #1 (low byte)
04FD --> Sound Timer #2 (low byte)
04FE --> Sound Timer #1 (high byte)
04FF --> Sound Timer #2 (high byte)
0500 --> USR start address (segment)
0501 --> USR start address (PC low)
0502 --> USR start address (PC high)
 .
 .
 .
0527 --> Keyboard IRQ (0)
0528 --> Keyboard IRQ (1)
0529 --> Keyboard IRQ (2)
052A --> Keyboard IRQ (3)
052B --> Keyboard IRQ (4)
052C --> Keyboard IRQ (5)
052D --> Keyboard IRQ (6)
052E --> Keyboard IRQ (7)
052F --> Keyboard IRQ (8)
0530 --> Keyboard IRQ (9)
 .
 .
 .
053D --> Character sets
         0: PC8X8
         1: COMPUTER8X8
         2: CIRILL8X8
 .
 .
 .
0543 --> SHIFT flag byte
```

```
          0. bit: Left SHIFT key is pressed (1) / not pressed (0)
          1. bit: Right SHIFT key is pressed (1) / not pressed (0)
          2. bit: Left CTRL key is pressed (1) / not pressed (0)
          3. bit: Right CTRL key is pressed (1) / not pressed (0)
          4. bit: Left ALT key is pressed (1) / not pressed (0)
          5. bit: Right ALT key is pressed (1) / not pressed (0)
 .
 .
 .
0548 --> Auto scroll down flag (0: off; >0: on)
0549 --> Section expand flag (0: no expand; >0: expand)
054A --> 0: no need section; >0: need section;
054B --> 0: BASIC; 255: Monitor
 .
 .
 .
055D --> Key1: SYS52650: REM SECRET CODE
 .       (32bytes)
 .
 .
057D --> Key2: DLOAD"
 .       (32bytes)
 .
 .
059D --> Key3: DIRECTORY
 .       (32bytes)
 .
 .
05BD --> Key4: SCNCLR
 .       (32bytes)
 .
 .
05DD --> Key5: DSAVE"
 .       (32bytes)
 .
 .
05FD --> Key6: RUN
 .       (32bytes)
 .
 .
061D --> Key7: LIST
 .       (32bytes)
 .
 .
063D --> Key8: HELP
 .       (32bytes)
 .
 .
065D --> Ethernet setup
         0.bit: No setup(0)/Setup(1)
         1.bit: TCP(0)/UDP(1)
         2.bit: no send(0)/send(1)
         3.bit: no received(0)/received(1)
         4.5.6.7.bits: Ethernet card number
065E --> Ethernet port number (low byte)
065F --> Ethernet port number (high byte)
```

```
0660 --> Receive buffer for Ethernet
 .       (256bytes)
 .
 .
0761 --> IP address (the 1st number)
0762 --> IP address (the 2nd number)
0763 --> IP address (the 3rd number)
0764 --> IP address (the 4th number)
0765 --> IP address (the 5th number)
0766 --> IP address (the 6th number)
0767 --> IP address (the 7th number)
0768 --> IP address (the 8th number)
0769 --> unused bytes (124bytes)
 .
 .
 .
07E5 --> Screen bottom
07E6 --> Screen top
07E7 --> Screen left
07E8 --> Screen right
 .
 .
 .
07EA --> 0: insert off; 255: insert on
07EB --> Flag for TheEnd (1: Reset; 2:Quit;)
07EC --> Flag for Mouse (0:off; >0:on;)
07ED --> Mouse button down (1:left; 2:middle; 4:right)
07EE --> Mouse X coordinate (low)
07EF --> Mouse X coordinate (high)
07F0 --> Mouse Y coordinate (low)
07F1 --> Mouse Y coordinate (high)
07F2 --> Accumulator register (SYS)
07F3 --> X register (SYS)
07F4 --> Y register (SYS)
07F5 --> Stack Pointer (SYS)
07F6 --> Virtual key code (low byte)
07F7 --> Virtual key code (high byte)
07F8 --> JOY flag (0: the 1st joystick; 1: the 2nd joystick;...)
07F9 --> JOY Button down number
07FA --> GamePad stick #1
         (0:base;1:N;2;NE;3:E;4:SE;5:S;6:SW;7:W;8:NW; 128...:fire)
07FB --> GamePad stick #2 (if exists)
07FC --> GamePad vibration effect (1: gun; 2: chainsaw)
07FD --> Pointer to the current memory segment (0..255)
         default is 0 --> DS = Data Segment
07FE --> Pointer to the current memory segment (0..255)
         default is 0 --> CS = Code Segment
07FF --> 0.bit: 0: CBM8x8; 1: PC8x8
         1.bit: 0: 16 color; 1: 256 color (text)
         2.bit: 0: normal mode; 1: invers mode
         3.bit: temporary bit for storing invers mode
                for previous cursor position
         4.bit: 0: normal mode; 1: blink mode
         5.bit: temporary bit for storing blink mode
                for previous cursor position
         6.bit: 0: graphic off; 1: graphic on
```

```
         7.bit: 0: 320x200bit graphics (bit/pixel);
                1: 320x200byte graphics (byte/pixel)
0800 --> Color attributes table for Text screen
         (40x25bytes: 17h: 1 for foreground 7 for background)
 .
 .
 .
0BE8 --> unused bytes (24bytes)
 .
 .
 .
0C00 --> Text screen
 .       40x25 = 1000 bytes
 .
 .
0FE8 --> Iteration flag (>0: need a goback)
0FE9 --> Stack pointer for iteration
0FEA --> Stack for iteration I. (GOSUB-RETURN)
 .       (256x5bytes)
 .
 .
14EA --> Stack pointer for Lengyel's form (STACK)
14EB --> Stack for Lengyel's form
 .       (256bytes)
 .
 .
15EB --> Queue pointer for Lengyel's form (QUEUE start)
15EC --> Queue pointer for Lengyel's form (QUEUE end)
15ED --> Queue for Lengyel's form
 .       (256bytes)
 .
 .
16EC --> Length of Lengyel's form of last calculation
16ED --> Lengyel's form of the last calculation
 .       (256bytes)
 .
 .
17EC --> unused bytes (18bytes)
 .
 .
 .
1800 --> Foreground color table for 256 color graphics mode
 .       (40x25bytes)
 .
 .
1BE8 --> unused bytes (24bytes)
 .
 .
 .
1C00 --> Background color table for 256 color graphics mode
 .       (40x25bytes)
 .
 .
1FE8 --> unused bytes (24bytes)
 .
 .
```

```
 .
2000 --> High Resolution Graphics Area
 .       320x200bits = 8000bytes
 .
 .
3F40 --> Stack pointer for iteration II.
3F41 --> Stack for iteration II. (TO numbers)
 .       (256x4bytes)
 .
 .
4341 --> Stack pointer for iteration III.
4342 --> Stack for iteration III. (STEP numbers)
 .       (256x4bytes)
 .
 .
4742 --> Stack pointer for iteration IV.
4743 --> Stack for iteration IV. (FOR-NEXT-VARIABLES)
 .       (256x5bytes)
 .
 .
4C43 --> Screen refreshing attributes
 .       (40x25bit=125bytes)
 .
 .
4CC0 --> Last command in embedded programs (3-PLUS-1)
 .       (40bytes)
 .
 .
4CE8 --> unused bytes (32962bytes)
 .
 .
 .
CDAA --> Secret code (address, e-mail and phone number of the author)
         (94bytes)
 .
 .
 .
CE08 --> unused bytes (6bytes)
 .
 .
 .
CE0E --> 0x40 (RTI) (100/1second)
CE0F --> unused bytes (496bytes)
 .
 .
 .
D000 --> CBM Characters (8x8)
 .       (256x8bytes)
 .
 .
D800 --> PC Characters (8x8)
 .       (256x8bytes)
 .
 .
E000 --> Foreground color table for 256 color text mode
 .       (40x25bytes)
```

```
 .
 .
E3E8 --> unused bytes (24bytes)
 .
 .
 .
E400 --> Background color table for 256 color text mode
 .       (40x25bytes)
 .
 .
E7E8 --> unused bytes (24bytes)
 .
 .
 .
E800 --> Invers attribute for each character
 .       (40x25bit = 125bytes)
 .
 .
E87D --> Frame Color
E87E --> Foreground color for graphics
E87F --> Background Color for graphics
E880 --> Blink attribute for each character
 .       (40x25bit = 125bytes)
 .
 .
E8FD --> Current section
E8FE --> Section remarkers
 .       (25bytes)
 .
 .
E917 --> User typed text after pressing Enter
 .       (1000+1bytes)
 .
 .
ED00 --> The length of the machine code (low byte)
ED01 --> The length of the machine code (high byte)
ED02 --> User typed command or program machine code for interpreter
 .       (4094bytes)
 .
 .
FD00 --> Input/Output area
 .
 .
 .
FD30 --> Keyboard scan --> FF08
 .
 .
 .
FD3F --> FD30 --> FF08
 .
 .
 .
FEFE --> Low byte of pointer in the previous machine code
FEFF --> High byte of pointer in the previous machine code
FF00 --> The first counter (low byte)
FF01 --> The first counter (high byte)
```

```
FF02 --> The second counter (low byte)
FF03 --> The second counter (high byte)
FF04 --> The third counter (low byte)
FF05 --> The third counter (high byte)
FF06 --> Screen mode#1
         0..2.bit: Y-scroll
            3.bit: text screen --> 0=24 rows; 1=25 rows
            4.bit: screen on/off (0=off)
            5.bit: 0=text; 1=graphics
            6.bit: text screen --> 0=16 colors; 1=256 colors
FF07 --> Screen mode#2
         0..2.bit: X-scroll
            3.bit: text screen --> 0=38 columns; 1=40 columns
            4.bit: double pixels
                   (four colors in a character)
                   TEXT:
                   00: color of FF15
                   01: color of FF16
                   10: color of FF17
                   11: color of 0800 or E000
                   GRAPHICS (bit/pixel):
                   00: color of FF15
                   01: color of FF16
                   10: color of FF17
                   11: color of FF18
FF08 --> Keyboard or Joystick
         KEYBOARD
         1.: write a byte to FD30 (eg.: 11111101b=FDh)
         2.: write a byte (<127) to FF08
         3.: read FF08 (keyboard scan)
         JOYSTICK
         1.: write a byte (!joynum) to FF08 (eg.: 11111110b=FEh)
         2.: read FF08 (joystick status --> 07FA)
FF09 --> IRQ (1=normal; 0=interrupt)
         3.bit: the first counter
         4.bit: the second counter
         6.bit: the third counter
         7.bit: CPU IRQ, if you set FF0A
FF0A --> IRQ enabled (1=enabled; 0=disabled)
         3.bit: the first counter
         4.bit: the second counter
         6.bit: the third counter
 .
 .
 .
FF0C --> Cursor position --> (0x00CA and 0x00CD)
         0.bit: 9.bit of FF0D
         1.bit: 10.bit of FF0D
FF0D --> Cursor position (low byte) --> (0x00CA and 0x00CD)
FF0E --> Frequency for sound generator #1 (low byte)
FF0F --> Frequency for sound generator #2 (low byte)
FF10 --> Frequency for sound generator #2 (high byte)
FF11 --> Sound generators
         0..3 bit: volume (*512)
         4. bit:
                0: Turn OFF sound generator #1
```

```
                1: Turn ON sound generator #1
          5. bit:
                0: Turn OFF sound generator #2
                1: Turn ON sound generator #2
          6. bit:
                0: Noise OFF
                1: Noise ON
          7. bit:
                0: DA OFF
                1: DA ON
FF12 --> Frequency for sound generator #1 (high byte)
FF13 --> Character table address
          0..1.bit: unused
          2..7.bit: new address of character table
FF14 --> Graphical screen address
          0..2.bit: unused
          3..7.bit: new address of graphical screen
FF15 --> 0086 --> Background color for text
FF16 --> 0085 --> Foreground color for text
FF17 --> E87F --> Background color for graphic
FF18 --> E87E --> Foreground color for graphic
FF19 --> E87D --> Frame color
FF1A --> Special effects
          0. bit: 0: Light off; 1: Light on;
          1. bit: 0: "Sun" off; 1: "Sun" on; (Light source on/off)
          2. bit: 0: Snowfall off; 1: Snowfall on;
FF1B --> Ambient light (1.byte): /255 = 0.0 ... 1.0 (Red)
FF1C --> Ambient light (2.byte): /255 = 0.0 ... 1.0 (Green)
FF1D --> Ambient light (3.byte): /255 = 0.0 ... 1.0 (Blue)
FF1E --> Diffuse light (1.byte): /255 = 0.0 ... 1.0 (Red)
FF1F --> Diffuse light (2.byte): /255 = 0.0 ... 1.0 (Green)
FF20 --> Diffuse light (3.byte): /255 = 0.0 ... 1.0 (Blue)
FF21 --> Specular light (1.byte): /255 = 0.0 ... 1.0 (Red)
FF22 --> Specular light (2.byte): /255 = 0.0 ... 1.0 (Green)
FF23 --> Specular light (3.byte): /255 = 0.0 ... 1.0 (Blue)
FF24 --> Light position (X: sign: 0:+ ; 1:- ;)
FF25 --> Light position (X: low byte)
FF26 --> Light position (X: high byte)
          /64 = 0.0 ... 1024.0
FF27 --> Light position (Y: sign: 0:+ ; 1:- ;)
FF28 --> Light position (Y: low byte)
FF29 --> Light position (Y: high byte)
          /64 = 0.0 ... 1024.0
FF2A --> Light position (Z: sign: 0:+ ; 1:- ;)
FF2B --> Light position (Z: low byte)
FF2C --> Light position (Z: high byte)
          /64 = 0.0 ... 1024.0
FF2D --> Number of snowflakes (0..255)
 .
 .
 .
FF3E --> Segment of character table (0 is default)
FF3F --> Segment of graphic screen (0 is default)
 .
 .
 .
```

```
      FFD2 --> Microcode: Output to channel
               (send a character to the cursor position)
       .
       .
       .
      FFF6 --> Microcode: Soft reset (pressing F11)
       .
       .
       .
      FFFE --> Microcode: Reboot the computer
      FFFF --> Microcode: Shutdown the computer
[001]:0000 --> TurboVision 4 BASIC Area (13107200bytes = 12800KiB = 12.5MiB)
  .    .
  .    .
  .    .
[023]:0000 --> Database (10000*25*38=9500000bytes)
  .    .
  .    .
  .    .
[168]:0000
       .
       .
       .
      F55F --> End of Database
      F560 --> Number of fields (1..25)
      F561 --> Fields name(35) and length(1..38)
       .       (25*(35+1)=25*36=900 bytes)
       .
       .
      F8E4 --> End of fields name and length
      F8E5 --> Fields counter (from 1 to "number of fields" * 2)
      F8E6 --> Last modified record number 1 .. 10 000 (low byte)
      F8E7 --> Last modified record number 1 .. 10 000 (high byte)
      F8E8 --> Current field number (1..25) for RC
      F8E9 --> Current position of search (4.byte)
      F8EA --> Current position of search (3.byte)
      F8EB --> Current position of search (2.byte)
      F8EC --> Current position of search (1.byte)
      F8ED --> PIck field number (1..25)
      F8EE --> HIGHRC (low byte)
      F8EF --> HIGHRC (high byte)
      F8F0 --> PI-BOTTOM
       .       (40bytes)
       .
       .
      F917 --> End of PI-BOTTOM
      F918 --> PI-TOP
       .       (40bytes)
       .
       .
      F93F --> End of PI-TOP
      F940 --> DiskSort field one
      F941 --> DiskSort field two
      F942 --> DiskSort field three
      F943 --> DiskSort swap area
       .       (25*38=950bytes)
```

```
       .
       .
      FCF8 --> End of DiskSort area
      FCF9 --> unused bytes (775bytes)
       .
       .
       .
[169]:0000 --> Spreadsheet Numbers (1000*25*4=100000bytes)
  .    .
  .    .
  .    .
[170]:0000
       .
       .
       .
      869F --> End of Spreadsheet Numbers
      86A0 --> Number or text flags (1000*25=25000bytes)
               0.bit: 0-off; 1-on
               1.bit: 0-NUMBER; 1-TEXT
               2.bit: 0-right align; 1-left align
               3.bit: 0-no formula; 1-FORMULA
       .
       .
       .
      E847 --> End of NoT flags
      E848 --> Column
      E849 --> Row (low byte)
      E84A --> Row (high byte)
      E84B --> X
      E84C --> Y
      E84D --> Manual/Auto (0:MANU; 1:AUTO)
      E84E --> Compiled code for a formula
       .       (128bytes)
       .
       .
      E8CD --> End of compiled code for a formula
      E8CE --> Solution of a formula
       .       (128bytes)
       .
       .
      E94D --> End of solution of a formula
      E94E --> unused bytes (5810bytes)
       .
       .
       .
[171]:0000 --> Spreadsheet Text (1000*25*36=900000bytes)
  .    .
  .    .
  .    .
[184]:0000
       .
       .
       .
      BB9F --> End of Spreadsheet Text
      BBA0 --> unused bytes (17504bytes)
       .
```

```
       .
       .
[185]:0000 --> Spreadsheet Formula (1000*25*36=900000bytes)
  .    .
  .    .
  .    .
[198]:0000
       .
       .
       .
      BB9F --> End of Spreadsheet Formula
      BBA0 --> unused bytes (17504bytes)
       .
       .
       .
[199]:0000 --> Word processor (1000*80=80000bytes)
  .    .
  .    .
  .    .
[200]:0000
       .
       .
       .
      387F --> End of text
      3880 --> Tabulators (80bit=10bytes)
       .
       .
       .
      3889 --> End of Tabulators
      388A --> Pointers (1000bit=125bytes)
       .
       .
       .
      3906 --> End of Pointers
      3907 --> Wordprocessor command {reverse} (1000*80bit=80000bit=10000bytes)
       .
       .
       .
      6016 --> End of command flags
      6017 --> Clipboard (484*80=38720bytes)
       .
       .
       .
      F756 --> End of Clipboard
      F757 --> Command line puffer (52*40+1=2081bytes)
       .
       .
       .
      FF77 --> End Command line puffer
      FF78 --> Column
      FF79 --> Row (low byte)
      FF7A --> Row (high byte)
      FF7B --> Last row
      FF7C --> Temporary storage for cursor X
      FF7D --> Temporary storage for cursor Y
      FF7E --> Temp char for RightScroll
```

```
      FF7F --> Temp Column: WP_X
      FF80 --> Temp Row: WP_Y low
      FF81 --> Temp Row: WP_Y high
      FF82 --> Temp flag for insert line (0-normal)
      FF83 --> unused bytes (125bytes)
       .
       .
       .
      FFFF --> Command mode
                 0 --> normal
                 1 --> Command line mode
                 2 --> CM (CLEAR ALL Y/N?)
                 3 --> SF (SAVE FILE:)
                 4 --> SF (REPLACE Y/N?)
                 5 --> LF (LOAD FILE:)
                 6 --> LF (NO FILE! :PRESS RETURN
                 7 --> SR (SEARCH:)
                 8 --> SR (CONTINUE Y/N?)
                 9 --> RE (SEARCH:)
                10 --> RE (BECOMES:)
                11 --> RE (CONTINUE Y/N?)
                12 --> RE (REPLACE Y/N?)
                13 --> PRESS ENTER
                14 --> ENTER NUMBER OF FIELDS 1..25
                15 --> ENTER FIELD NAME (1..35) FIELD # nn;
                16 --> ENTER FIELD LENGTH 1..38 FIELD # nn;
                17 --> ARE YOU SURE Y/N?
                18 --> New database record editor
                19 --> DF (DELETE FILE:)
                20 --> MF Merge File --> (LOAD FILE:)
                21 --> RV ReView
                22 --> PI PIck --> BOTTOM:
                23 --> PI --> TOP:
[201]:0000 --> Variables (3276800bytes = 3200KiB = 3.125MiB)
  .    .
  .    .
  .    .
[251]:0000 --> TrueColor Graphics Area (Red component)
       .       (320x200bytes)
       .
       .
      FA00 --> Mouse shape (16x16) for TrueColor (Red)
       .       256bytes
       .
       .
      FB00 --> unused bytes (1280bytes)
       .
       .
       .
[252]:0000 --> TrueColor Graphics Area (Green component)
       .       (320x200bytes)
       .
       .
      FA00 --> Mouse shape (16x16) for TrueColor (Green)
       .       256bytes
       .
```

```
       .
      FB00 --> unused bytes (1280bytes)
       .
       .
       .
[253]:0000 --> TrueColor Graphics Area (Blue component)
       .       (320x200bytes)
       .
       .
      FA00 --> Mouse shape (16x16) for TrueColor (Blue)
       .       256bytes
       .
       .
      FB00 --> unused bytes (1280bytes)
       .
       .
       .
[254]:0000 --> TrueColor Graphics Area (Alpha component)
       .       (320x200bytes)
       .
       .
      FA00 --> Mouse shape (16x16) for TrueColor (Alpha)
       .       256bytes
       .
       .
      FB00 --> unused bytes (1280bytes)
       .
       .
       .
[255]:0000 --> High Resolution Graphics Area
       .       320x200bytes - 256 colors
       .
       .
      FA00 --> RGBA colors for colors
       .       256x4bytes
       .
       .
      FE00 --> Mouse shape (16x16) for text, bit/pixel and byte/pixel
       .       256bytes
       .
       .
      FF00 --> unused bytes (256bytes)
       .
       .
       .
      FFFF --> End of memory
```

Chapter 10

Bibliography

- Bencsikné Takács Márta: Feladatgyűjtemény, 1986.
- Commodore Újság: Az Országos Commodore Egyesület lapja
- Commodore Plus/4 - A beépített programok kezelése - Grafika, Számviteli tömb, Szövegszerkesztő, Adatbázis kezelő - NOVOTRADE - 1986.
- Commodore Plus/4 Felhasználói kézikönyv - NOVOTRADE, 1987.
- Csákány Antal - Dr. Vajda Ferenc: Játékok számítógéppel, Műszaki Könyvkiadó, Budapest, 1985.
- Dr. Kocsis András: TV-BASIC, Számítástechnika-alkalmazási Vállalat, Budapest, 1984.
- Google
- Könözsy László - Gál István: Csupa szuperjáték C-PLUS/4, C-16, C-116 számítógépekre, Műszaki könyvkiadó, Budapest, 1989.
- Mikrovilág Magazin
- Németh Ferenc: Számítástechnika fakultáció 7-8. osztály, Pedagógiai Műhely - Pest Megyei Pedagógiai Intézet, 1989.
- Pál Zsuzsanna- Révbíró Tamás: Hetedhét, Novotrade, 1986.