VIDEOTON TV-COMPUTER PROGRAMMING GUIDE

version: February 9, 2023.

TABLE OF CONTENT

SHORT STORY OF TV-COMPUTER

The Videoton TV-Computer (aka TVC) was released in 1985, Hungary.

In **1980**, the development department at **VIDEOTON** proposed the design of a home computer, but the company's management rejected the idea.

Then in the spring of **1984**, at a chess computer exhibition in Budapest, VIDEOTON's commercial director met a representative of a British company, **Intelligent Software Ltd** (aka **ISL**), and they agreed to develop a microcomputer.

The ISL sold VIDEOTON the designs of an early prototype of their *Enterprise* computer, which VIDEOTON's engineers improved and modified to be manufactured from components available in Hungary. ISL developer *Bruce Tanner* laid the foundations of the computer's operating system with BASIC and the VT-DOS system, polished and completed by VIDEOTON engineers.

Interestingly, Bruce developed the TV-Computer BASIC and the VT-DOS operating system on an IBM 5150 PC, using a Z80 card called "Baby Blue."

The first units were produced at the end of **1985**, but not until **1986** that the TV-Computer was available in larger quantities in Hungarian shops for consumers. It was cheaper than a Commodore 64 or any other microcomputer available in Hungary, except the ZX 81.

At first months, there was very little software for the TV-Computer, so VIDEOTON signed a contract with *Novotrade*, a Hungarian company, to develop and distribute TV-Computer games and school education software. Novotrade was already developing games for several microcomputers, primarily the *Commodore 64* and *ZX Spectrum*. *Novotrade* sold its developed games mainly in England. Hungarian software developers created these games.

But they needed to make a lot of games quickly, which they didn't have enough staff to do, so they also released games made at home by the high school kids who went to the TV-Computer Club.

In **1987-1988** Novotrade imported cheap *Commodore Plus/4* and *Commodore 16* computers, which had failed in other markets. These computers were bought very cheaply then sold very cheaply in Hungary. VIDEOTON could not compete with this price, so they stopped production in **1989-1990**, and Novotrade stopped releasing new games and software for VIDEOTON TV-Computers in **1990**.

HARDVER SPECIFICATIONS

Models: TV-Computer 32k, TV-Computer 64k, TV-Computer 64k+
CPU: Z80 @ 3.125 MHz
Memory: 32 KB or 64 KB RAM
Video RAM: separated 16 KB in the 32k and 64k models, and 4 x 16 KB in the 64k+ model
ROM: 20 KB with BASIC, OS I/O functions, and the charset
Keyboard: 66 keys (*QWERTZ Hungarian layout*)+ internal joystick
Graphic modes:

- **512 x 240** pixels, **2** colors
- **256 x 240** pixels, **4** colors
- **128 x 240** pixels, **16** colors

Screen controller: CRTC 6845
Screen frequency: 50 Hz
Colors: fix **16-colors** palette with 2 black colors (s*o actually **15 colors***)
Screen memory: bitmap-like – every pixel contains color information too
Screen size: with 240 lines (default): **15 360** bytes; with 256 lines (optional): **16 384** bytes
Character mode: none (*chars are drawn graphically by BASIC*)
Definable characters: yes, (*definable character codes: 128-233*)
Sound: 1 channel, squarewave sound with 6 octaves and 4-bit volume
Sprites: no hardware sprites
OS:

- OS in **ROM** with **BASIC** and **I/O** (v1.2, v2.0, v2.1, v2.2)
- **VT-DOS** *(Disk Operating System)* which is MS-DOS compatible at the instruction level
- **UPM - CP/M 2.2** compatible at the program level

Floppy drive: external PC DOS compatible Floppy Disk File System with 360 KB and 720 KB disk sizes
Ports:

- 4 expansion slots
- 1 cartridge port
- 2 joystick ports
- 1 printer port

Conectors:

- 1 video connector
- 1 RGB connector
- 1 VHF/UHF antenna connector
- 2 tape connectors

Switches:

- black and white switch
- reset swith

Expansion cards *(original cards from 1985-1988)*:

- 32 KB RAM extension for TV-Computer 32k model only
- Floppy Interface Card
- Serial Line Card
- Parallel Card

GRAPHIC MODES

The TV-Computer has three graphical modes

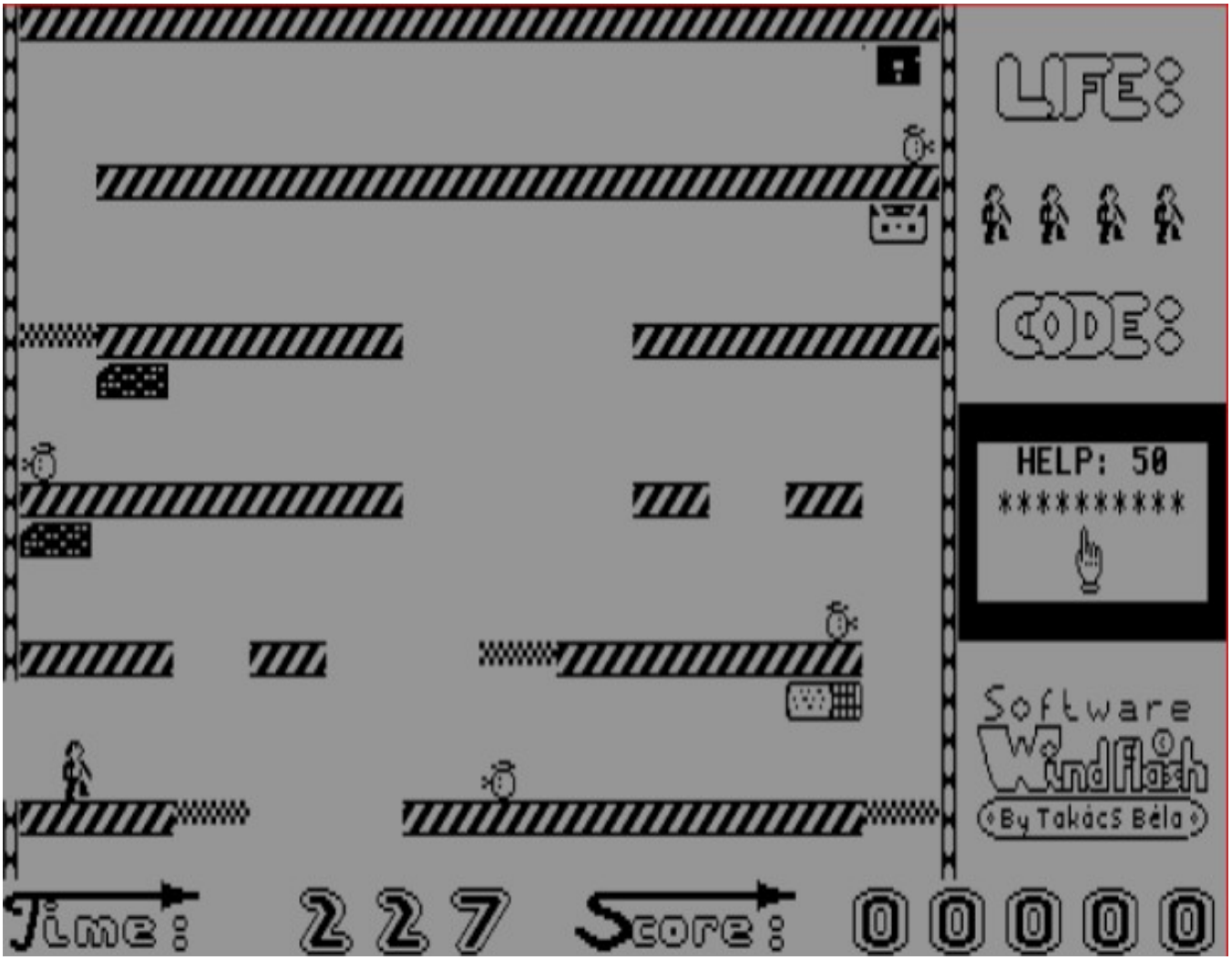
Graphic mode	Resulotion	Colors	Text	Pixels / byte	Screen size
Graphics 2	**512 x 240 pixels**	**2 colors**	**64 x 24 chars**	**8**	**(512 x 240) / 8**
Graphics 4	**256 x 240 pixels**	**4 colors**	**32 x 24 chars**	**4**	**(256 x 240) / 4**
Graphics 16	**128 x 240 pixels**	**16 colors**	**16 x 24 chars**	**2**	**(128 x 240) / 2**

The screen size is 15 360 bytes in all graphic modes.

A game in Graphics 2 mode

*Resolution **512 x 240** pixels. The 2 colors of this graphics mode can be any 2 colors from the 16-color palette.*

Reaktor (Reactor), 1987



A game in Graphics 4 mode

*Resolution **256 x 240** pixels. The 4 colors of this graphics mode can be any 4 colors from the 16-color palette. A pixel on the screen can be any color from the 4 colors selected in the palette.*

roHAMM (Attack), 1989

A game in Graphics 16 mode

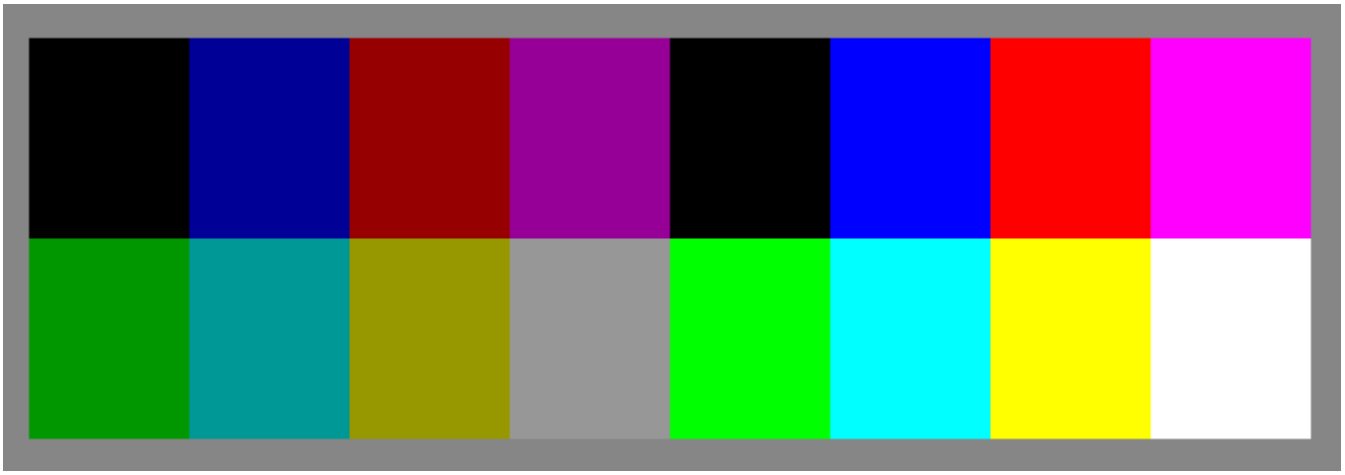
*Resolution **128 x 240** pixels. All colors from the 16-color palette can be used in this graphics mode. A pixel on the screen can be any color from the 16-color palette.*

Bomberman Megablast, 2021

COLORS

I	G	R	B	Color	GRAPHICS 16 Color codes in BASIC	GR. 2, 4 paletta hex.	GR. 2, 4 paletta dec.	Border hex.	Border dec.
0	0	0	0	Black	0	00	0	00	0
0	0	0	1	Dark blue	1	01	1	02	2
0	0	1	0	Dark red	2	04	4	08	8
0	0	1	1	Dark magenta	3	05	5	0A	10
0	1	0	0	Dark green	4	10	16	20	32
0	1	0	1	Dark cyan	5	11	17	22	34
0	1	1	0	Dark yellow	6	14	20	28	40
0	1	1	1	Gray	7	15	21	2A	42
1	0	0	0	Black	8	40	64	80	128
1	0	0	1	Blue	9	41	65	82	130
1	0	1	0	Red	10	44	68	88	136
1	0	1	1	Magenta	11	45	69	8A	138
1	1	0	0	Green	12	50	80	A0	160
1	1	0	1	Cyan	13	51	81	A2	162
1	1	1	0	Yellow	14	54	84	A8	168
1	1	1	1	White	15	55	85	AA	170

The palette of the TV-Computer



MEMORY MAPPING

The default memory mapping when the TV-Computer started:

Page 0: **$0000-3FFF** -> user RAM (**U0** - but **$0000-$19EE** ***is used by the system***)
Page 1: **$4000-7FFF** -> user RAM (**U1** – user memory, free to use)
Page 2: **$8000-BFFF** -> user RAM (**U2** - user memory, free to use → *64 and 64+ models*)
Page 3: **$C000-FFFF** -> system ROM (**SYS** - BASIC and system functions)

When the VIDEORAM is ON:

Page 0: **$0000-3FFF** -> user RAM (**U0** - but **$0000-$19EE** ***is used by the system***)
Page 1: **$4000-7FFF** -> user RAM (**U1** - user memory, free to use)
Page 2: **$8000-BFFF** -> **VIDEORAM**
Page 3: **$C000-FFFF** -> system ROM (**SYS** - BASIC and system functions)

Assembly example:

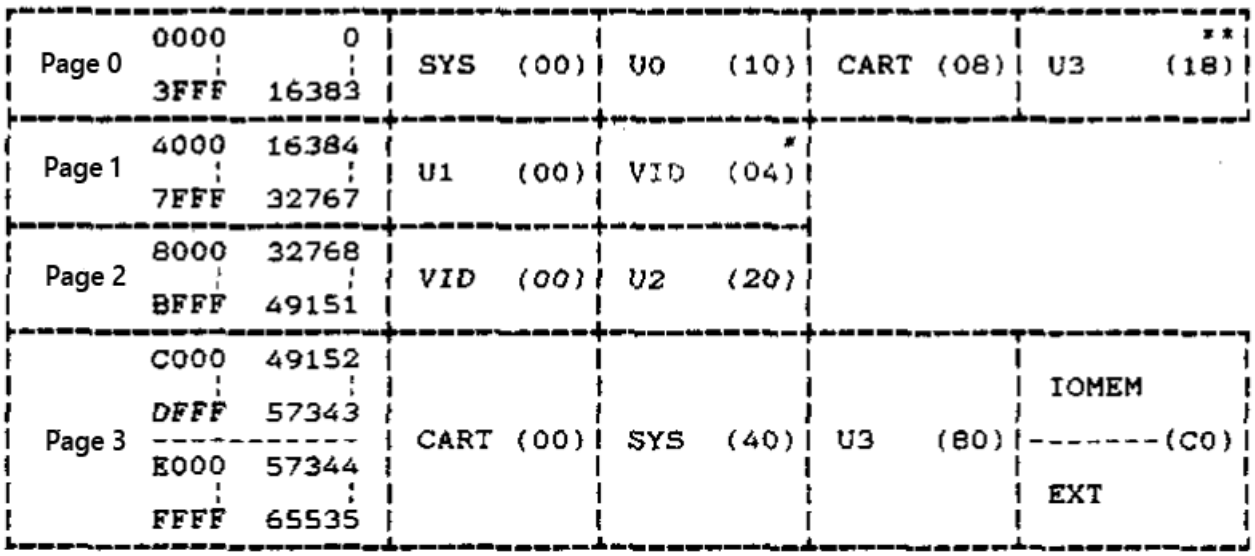
```
SET_VID_ON      ld      A,$50       ; memory mapping: U0, U1, VID, SYS
                ld      ($3),A      ; save value to P_SAVE system variable
                out     ($2),A      ; send value to set memory mapping
```

When the entire 64k memory is paged:

Page 0: **$0000-3FFF** -> user RAM (**U0** - but **$0000-$19EE** ***is used by the system***)
Page 1: **$4000-7FFF** -> user RAM (**U1** - user memory, free to use)
Page 2: **$8000-BFFF** -> user RAM (**U2** - user memory, free to use → *64 and 64+ models*)
Page 3: **$C000-FFFF** -> user RAM (**U3** - user memory, free to use → *64 and 64+ models*)

Assembly example:

```
SET_64K_RAM     ld      A,$B0       ; memory mapping: U0, U1, U2, U3
                ld      ($3),A      ; save value to P_SAVE system variable
                out     ($2),A      ; send value to set memory mapping
```



SYSTEM VIDEO CALLS

These system calls are in **ROM** and are used by **BASIC**. They are not the fastest solutions, as they support all three graphical modes, but they are suitable for initialization or non-speed-critical tasks.

Function	Code	Params in register(s)	Error	Assembly example
VMODE set video / graphic mode	**04h**	**C**: graphic mode ***0**: Graphics 2 (512x240)* ***1**: Graphics 4 (256x240)* ***2**: Graphics 16 (128x240)*	**0F7H** invalid graphics mode	ld **C**,2 ; param: graphics 16 rst $30 ; system call db **$4** ; function code
PAL set palette	**0Ch**	**DE**: point to palette data *4 byte data with palette color codes*	-	ld **DE**,palette ; palette address rst $30 ; system call db **$C** ; function code
CLS clear screen	**05h**	-	-	rst $30 ; system call db **$5** ; function code
BTEXT set character position	**03h**	**B**: column (**1-16;1-32;1-64**) **C**: line (**1-24**)	**0F9H** invalid position	ld **BC**,$0107 ; col.:1; line: 7 rst $30 ; system call db **$3** ; function code
VID_CHOUT write a char to current pos.	**01h**	**C**: character code	-	ld **C**,$2A ; “*” char code rst $30 ; system call db **$1** ; function code

VID_BKOUT
write text to current pos.
	02h	**DE**: text memory address
		BC: text length
			-	ld **DE**,text ; text label
				ld **BC**,$1B ; text length
				rst $30 ; system call
				db **$2** ; function code

BABS
set pixel pos. for writing char/text or drawing line
	06h	**DE**: X (*horizontal*) position
		BC: Y (*vertical*) position
		*X: **0 – 1023** (BASIC pos.)*
		*Y: **0 – 959** (BASIC pos.)*
			0F9H
			invalid position
				ld **DE**,$CB ; X position
				ld **BC**,$64 ; Y position
				rst $30 ; system call
				db **$6** ; function code

BREL set relative pixel position for text or line	**07h**	**DE**: horizontal displacement **BC**: vertical displacement	**0F9H** invalid position	d **DE**,$CB ; horiz. disp. ld **BC**,$64 ; vert. sisp. rst $30 ; system call db **$7** ; function code
BON pen ***On***	**08h**	*and put a pixel to current position*	-	rst $30 ; system call db **$8** ; function code
BOFF pen ***Off***	**09h**	Then **BABS** and **BREL** do not draw a line.	-	rst $30 ; system call db **$9** ; function code
FILL fill from current pos.	**0Ah**	You can set the fill position by **BABS** or **BREL** calls.	-	rst $30 ; system call db **$A** ; function code

DEFC
define character
	0Bh	**C**: char code (**128 - 223**)
		DE: character matrix data – 10 bytes (1 byte = 1 line in char matrix, 1 bit = 1 pixel)
			0F8H
			invalid character
				ld **C**,$80 ; char. code
				ld **DE**,char_data
				rst $30 ; system call
				db **$B** ; function code

SCREEN / VIDEORAM

The screen starting address is **$8000** when the **VIDEORAM** is mapped to **page #2**.

By default, the screen contains **240 lines**; each line is **64 bytes** in all three resolutions.

The screen is sequential in the VIDEORAM.

VIDEORAM

LINE no.	0	1	2	3	- - -	61	62	63	CRTC CHAR. LINE
0	8000	8001	8002	8003	- - -	803D	803E	803F	
1	8040	8041	8042	8043	- - -	807D	807E	807F	
2	8080	8081	8082	8083	- - -	80BD	80BE	80BF	0
3	80C0	80C1	80C2	80C3	- - -	80FD	80FE	80FF	
4	8100	8101	8102	8103	- - -	813D	813E	813F	
5	8140	8141	8142	8143	- - -	817D	817E	817F	
6	8180	8181	8182	8183	- - -	81BD	81BE	81BF	1
7	81C0	81C1	81C2	81C3	- - -	81FD	81FE	81FF	
8	8200	8201	8202	8203	- - -	823D	823E	823F	
9	8240	8241	8242	8243	- - -	827D	827E	827F	
10	8280	8281	8282	8283	- - -	82BD	82BE	82BF	2
11	82C0	82C1	82C2	82C3	- - -	82FD	82FE	82FF	
:	:	:	:	:		:	:	:	:
232	BA00	BA01	BA02	BA03	- - -	BA3D	BA3E	BA3F	
233	BA40	BA41	BA42	BA43	- - -	BA7D	BA7E	BA7F	
234	BA80	BA81	BA82	BA83	- - -	BABD	BABE	BABF	58
235	BAC0	BAC1	BAC2	BAC3	- - -	BAFD	BAFE	BAFF	
236	BB00	BB01	BB02	BB03	- - -	BB3D	BB3E	BB3F	
237	BB40	BB41	BB42	BB43	- - -	BB7D	BB7E	BB7F	
238	BB80	BB81	BB82	BB83	- - -	BBBD	BBBE	BBBF	59
239	BBC0	BBC1	BBC2	BBC3	- - -	BBFD	BBFE	BBFF	

Graphic mode	Resulotion	Colors	Pixels / byte	Screen size
Graphics 2	**512 x 240 pixels**	**2 colors**	**8**	**(512x240)/8**
Graphics 4	**256 x 240 pixels**	**4 colors**	**4**	**(256x240)/4**
Graphics 16	**128 x 240 pixels**	**16 colors**	**2**	**(128x240)/2**

The screen size is 15 360 bytes in all graphic modes. *If you switch on all the 256 lines, the screen size is **16 384** bytes.*

The content of a byte depends on the graphics mode.

1. Graphics 2 – 512 x 240 pixels with 2-color palette (any 2 colors from the 16-color palette)

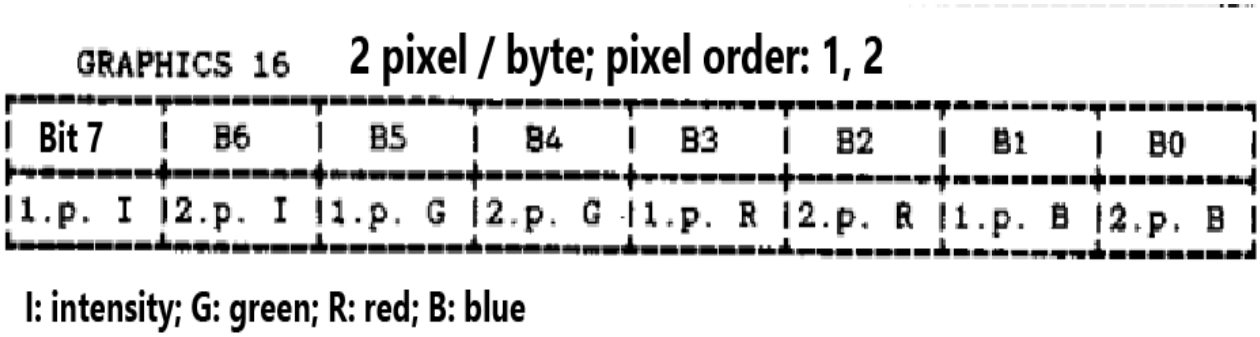
Each bit is 1 pixel in the byte. If the bit is 0, it displays the color 0 of the palette; if the bit is 1, it shows the color 1.

2. Graphics 4 – 256 x 240 pixels with 4-color palette (any 4 colors from the 16-color palette)

This is a bit more complicated. Here there are 2 bits per pixel, which can have a value of 0-3, indicating which color of the 4-color palette the pixel will be in. The table below shows you what this looks like.

Bit #7	Bit #6	Bit #5	Bit #4	Bit #3	Bit #2	Bit #1	Bit #0
Pixel #1 color bit #0	**Pixel #2** color bit #0	**Pixel #3** color bit #0	**Pixel #4** color bit #0	**Pixel #1** color bit #1	**Pixel #2** color bit #1	**Pixel #3** color bit #1	**Pixel #4** color bit #1

3. Graphics 16 – 128 x 240 pixels with 16 colors *(plain 16 colors)*



If the intensity bit is 0, it is the darker version of the color; if it is 1, it is the lighter one. Since intensity is a multiplier on RGB values, there are two blacks in the 16-color palette.

PUT IMAGE EXAMPLE

Basic information for putting a picture or a sprite to screen:

Screen memory starting address: **$8000** *(if VIDEORAM is set)*

Width of a line: **64 bytes** *(in all graphic modes)*

Screen height: **240 lines** *(0 - 239)*

Screen position: **$8000** + **Ypos** * **64** + **Xpos**

*In the screen position calculation, **Xpos** is the horizontal position in bytes (0 - 63), and **Ypos** is the vertical position in lines (0 - 239).*

The bytes of the image are pixels according to the graphics mode you are using. Graphics 4 mode is used in the example code below, which is the default after turning on the TV-Computer.

```
SET_VID_ON LD   A,$50              ; memory mapping: U0, U1, VID, SYS
           LD   ($3),A             ; put value to P_SAVE system variable
           OUT  ($2),A             ; send value to port #2
           ; init PUT_IMAGE
           LD   HL,IMAGE_DATA      ; HL → image data
           LD   DE,$8000+106*64+30 ; DE → screen pos.; line: 106; col: 30
           LD   A,28               ; A = image height in pixels
           CALL PUT_IMAGE          ; call put image subroutine
           RET                     ; return to BASIC (not necessary)
           ; PUT_IMAGE - params: HL:image, DE:address; A:image height
PUT_IMAGE  LD   BC,64              ; BC = 64 → one line on the screen is 64 bytes
           ; image width is 4 bytes, using 4 x LDI to put one line
           LDI                     ; copy 1 byte from (HL) to (DEL) and Dec(BC)
           LDI                     ; copy 1 byte from (HL) to (DEL) and Dec(BC)
           LDI                     ; copy 1 byte from (HL) to (DEL) and Dec(BC)
           LDI                     ; copy 1 byte from (HL) to (DEL) and Dec(BC)
           EX   DE,HL              ; switch DE <=> HL
           ADD  HL,BC              ; HL = HL + (64 -ImageWidth) → new line
           EX   DE,HL              ; switch back DE <=> HL
           DEC  A                  ; decrement remaining lines
           JP   NZ,PUT_IMAGE       ; if A > 0, then loop
           RET                     ; done, return to caller
           ; the image pixels data in bytes (16x28 pixel → 4x28 bytes)
IMAGE_DATA DB    48,0,0,192,112,176,208,224,112,240,240,224,112,240,240,224
           DB    48,48,192,192,96,240,240,96,112,240,240,224,112,48,192,224
           DB    48,48,192,192,48,240,240,192,112,192,48,224,112,224,112,224
           DB    48,176,208,192,16,192,48,128,2,112,224,4,7,56,193,12
           DB    7,12,3,14,15,15,15,15,9,14,7,9,120,15,15,225
           DB    112,14,7,224,1,15,15,8,32,0,0,64,48,240,240,192
           DB    16,240,240,128,0,224,112,0,0,14,7,0,3,14,7,12
           END
```

After running the above assembly example program, you will see this on the screen.

CRTC 6845

IMPORTANT! The *CRTC* uses the concept of ***special*** *characters* and *lines*. This is not the same as the characters in the texts.

One screen line is **64** *CRTC characters*, and the default 240 lines height screen contains **60** *CRTC lines*. The entire 256 lines height screen has **64** *CRTC lines*.

A *CRTC line* height is 4 pixels (256 / 64 = 4).

The CRTC registers

				Default	
Reg.	R/W	Unit	Description	Hex.	Dec.
R0	**W**	char	(characters / line) - 1	**63**	**99**
R1	**W**	char	Displayed chars / line	**40**	**64**
R2	**W**	char	Horizontal sync position - 1	**4B**	**75**
R3	**W**		Bit 0-3: horiz. sync. Chars; 4-7: vert. sync lines.	**32**	**50**
R4	**W**	char line	Bit 0-6: (all character lines / screen) - 1	**40**	**77**
R5	**W**	TV line	Bit 0-4: additional TV lines	**2**	**2**
R6	**W**	char line	Bit 0-6: displayed character lines / screen	**3C**	**60**
R7	**W**	char line	Bit 0-6: vertical sync position - 1	**42**	**66**
R8	**W**		**Bit 0-1**: interlace (0/1); **4-5**: *DISPTMC* **6-7**: char displacement.	**0**	**0**
R9	**W**	TV line	Bit 0-4: (TV lines / char. Lines) - 1	**3**	**3**
R10	**W**	TV line	Bit 0-4: cursor starting line no. in char; 5-6: enabled (00) / disabled (01)	**3**	**3**
R11	**W**	TV line	Bit 0-4: cursor ending line no.	**3**	**3**
R12	**R/W**		Bit 0-5: screen start address high byte	**0**	**0**
R13	**R/W**		screen start address low byte	**0**	**0**
R14	**R/W**		Bit 0-5: raster-interrupt position high byte	**0E**	**14**
R15	**R/W**		raster-interrupt position low byte	**FF**	**255**
R16	**R**		Bit 0-5: light pen position high byte		
R17	**R**		light pen position low byte		

You can **select** a ***CRTC register*** by the port **$70**.

You can **write or read** a ***CRTC register*** by the port **$71**.

Please check out the following assembly source code examples.

SET RASTRER-INTERUPT POSITION BY CRTC

By default, the Raster-interrupt position is line **#239** *(CRTC line 60)* and column (byte) **#63**.

An example of how to set the raster-interrupt position to the last byte of the line that passed in the parameter.

```
                    ; Param: HL - raster-IT position (CRTC line * 64)  - 1
SET_RASTER_IT_POS   ld      A,$E
                    out     ($70),A         ; select CRTC Reg. #14
                    ld      A,H
                    out     ($71),A         ; set position HIGH byte
                    ld      A,$F
                    out     ($70),A         ; select CRTC Reg #15
                    ld      A,L
                    out     ($71),A         ; set position LOW byte
                    ret
```

For example, set the raster-interrupt position 16 lines above the bottom of the screen.

```
                    ; Set the Raster-interrupt position to the last byte of line #240
                    ; One line is 64 bytes, and one CRTC line height is 4 pixels.
                    ld      HL,(50*64)-1
                    call    SET_RASTER_IT_POS
```

Accurate calculation of raster-interrupt

Ypos → raster-interrupt vertical position (0-239)
Xpos → raster-interrupt horizontal position (0-63)

CharNo = INT (**Ypos** /4)
TVLineNo = **Ypos** – (**CharNo*4)**
RasterPosAddr = **CharNo** * 64 + **Xpos**

Set CRTC registers for raster-interrupt:

R14 = INT (**RasterPosAddr** / 256) ; the RasterPosAddr high byte

R15 = **RasterPosAddr** - INT (**RasterPosAddr** / 256) ; the RasterPosAddr low byte

R10 = **TVLineNo**

R11 = TVLineNo

IMPORTANT! Never set Xpos to 0, as this may cause an error!

The best practice is to set Xpos to 63. This is the last character of the CRTC line.

SET SCREEN START POSITION BY CRTC

```
                ; Param: HL - screen_offset (value: 0-4095)
                ; The screen contains 64 CRTC lines and 64 bytes / line
                ; 64*64 = 4096
                ; if param HL = 0, the screen starts from the VIDEORAM 1st line
                ; if param HL = 256, the screen starts from the VIDEORAM 16th line
                ; if param HL = 640, the screen starts from the VIDEORAM 40th line
SET_SCREEN_START ld     A,$C
                out     ($70),A         ; select CRTC Reg. #12
                ld      A,H
                out     ($71),A         ; set screen position HIGH byte
                ld      A,$D
                out     ($70),A         ; select CRTC Reg #13
                ld      A,L
                out     ($71),A         ; set position LOW byte
                ret
```

If you continuously increase the starting address of the screen by one using the CRTC, it will result in a continuous vertical scroll. For each increase, the screen will move by 4 rows of pixels.

PORTS

Dec.	Hex.	R/W	Description
0	**0**	**W**	Border port to set border color; bit 1: Blue; 3: Red; 5: Green; 7: Intensity
1	**1**	**W**	Printer data out port (character to printer)
2	**2**	**W**	Bit 2-7: memory mapping (It **must be set** at memory address **02h** first!)
3	**3**	**W**	Bit 0-3: select keyboard line from matrix; 6-7: Slots 0-3 IOMEM mapping
4	**4**	**W**	Sound low byte
5	**5**	**W**	Bit 0-3: sound high byte; 4: sound disable(1); 5: sound IT enable(1); 6-7: tape control (6: left tape connector; 7 right tape connector)
6	**6**	**W**	0-1: graphic mode; 1-5: sound volume; 7: printer / STROBE
7	**7**	**W**	Clear raster/sound-interrupt *(any value)*
15	**0**	**W**	Bit 4-5: set VIDEORAM bank on 64k+ model; Value = (BANK – 1) * 16
16	**10**	**W**	Slot #0: serial line data
17	**11**	**W**	Slot #0: serial line USART mode select
32	**20**	**W**	Slot #1: serial line data
33	**21**	**W**	Slot #1: serial line USART mode select
48	**30**	**W**	Slot #2: serial line data
49	**31**	**W**	Slot #2: serial line USART mode select
64	**40**	**W**	Slot #3: serial line data
65	**41**	**W**	Slot #3: serial line USART mode select
80	**50**	**W**	Tape output signal
88	**58**	**R**	Read keyboard line from the keyboard matrix (first select on port #3)
89	**59**	**R**	0-3: Slot 0-3 irq req.; 4: raster/sound irq req.; 5: tape data; 6: B/W(0) / Color (1) mode; 7: printer ACK
90	**0**	**R**	Bit 0-1: Slot 0 ID; 2-3: Slot 1 ID; 4-5: Slot 2 ID; 6-7: Slot 3 ID ID 00: serial line; ID 10: floppy interface; ID 01: not used; ID 11: not spec.
91	**0**	**R**	Clearing the sound generator frequency divider for accurate timing
96	**60**	**W**	Palette color #0
97	**61**	**W**	Palette color #1
98	**62**	**W**	Palette color #2
99	**63**	**W**	Palette color #3
112	**70**	**W**	Bit 0-4: CRTC 6845 register selection (value: 0 - 17)
113	**71**	**R/W**	CRTC 6845 register data

Set the **Dark Blue, Cyan, Dark Red**, and **Yellow** colors palette in **Graphics 4** mode:

```
ld A,$1       ; A = Dark Blue color
out ($60),A   ; set Dark Blue color to Palette Color #0
ld A,$51      ; A = Cyan color
out ($61),A   ; set Cyan color to Palette Color #1
ld A,$4       ; A = Dark Red color
out ($62),A   ; set Dark Red color to Palette Color #2
ld A,$54      ; A = Yellow color
out ($63),A   ; set Yellow color to Palette Color #0
```

Set Border color to Dark Blue

```
ld A,$1         ; A = Dark Blue color
out ($0),A      ; set border color to Dark Blue
```

Set Graphics 16 mode

```
ld      B,2                 ; B = 2 → Graphics 16 mode code (0: Graphics 2; 1: Graphics 4)
ld      A,($0B13)           ; A = Port #6 mirror in the memory (system variable)
and     128+64+32+16+8+4    ; clear bit #0 and #1 → graphic mode bits
or      B                   ; set graphic mode to bit #0 and #1
ld      ($0B13),A           ; save value to port #6 mirror system variable
out     ($6),A              ; send value to port
```

Set Sound ON

```
ld      A,($0B12)           ; A = port #5 mirror from system variable
and     128+64              ; clear low 6 bits
or      $F                  ; enable sound sign (bit #4)
ld      ($0B12),A           ; set new value to port #5 mirror system variable
out     ($5),A              ; send new value to port #5
```

Sound

```
ld      HL,$E5D             ; HL = octave 4; note #A value (PITCH in BASIC)
ld      A,L                 ; A = note low byte
out     ($4),A              ; send low byte to SOUND port
ld      A,($0B12)           ; A = port #5 mirror from system variable
and     128+64+32+16        ; clear low 4 bits for the note high 4 bits
or      H                   ; A = port #5 value or note high 4 bits
out     ($5),A              ; send value to port #5
```

Set Sound Volume

```
ld      A,$F                ; A = Volume value (0-15)
sla     A                   ; shift bits to left
sla     A                   ; shift bits to left → A = volume on #2 - #5 bits.
ld      E,A                 ; E = A
ld      A,($0B13)           ; A = port #6 mirror system variable
and     128+64+2+1          ; clear #2 - #5 bits (volume bits)
or      E                   ; A = port #6 value or Volume value on #2-#5 bits
ld      ($0B13),A           ; set new value to port #6 mirror
out     ($6),A              ; send Volume to port #6
```

SYSTEM VARIABLES

SYSTEM variables on U0 (Page 0) memory *(length in bytes)*

hex addr.	dec.addr.	Length	Name	Decription
0003	**3**	**1**	**P_SAVE**	Memory mapping
0030	**48**	**8**	**OS_ENTRY**	Entry point of ROM functions (**rst $30**)
0038	**56**	**8**	**IT_ENTRY**	Entry point if Interrupt Handler
0040	**64**	**192**		IDs of the expansion cards
0B00	**2816**	**8**	**IN_TABLE**	Input assignment table
0B08	**2824**	**8**	**OUT_TABLE**	Output assignment table
0B10	**2832**	**1**	**INT_DES**	Interrupt served device
0B11	**2833**	**1**	**PORT_03**	Memory mirror of Port #3
0B12	**2834**	**1**	**PORT_05**	Memory mirror of Port #5
0B13	**2835**	**1**	**PORT_06**	Memory mirror of Port #6
0B14	**2836**	**1**	**SND_ACTIVE**	**FF**: sound is in progress
0B15	**2837**	**1**	**SND_IRQ**	**FF**: The new sound interrupts what is in progress
0B16	**2838**	**1**	**STOP_FLAG**	**FF:** CTRL+ESC pressed *(this stops the BASIC program)*
0B17	**2839**	**2**	**ST_LIMIT**	Lower limit of the STACK when FILL in progress
0B19	**2841**	**2**	**HI_MEM**	Highest memory address for BASIC
0B1b	**2843**	**1**	**P3RAM**	**0**: ***Page 3*** (U3) RAM test OK; **1**: ***Page 3*** RAM test fault
0B1C	**2844**	**1**	**EX_DEF**	Default assignment of expansion card
0B1D	**2845**	**2**	**INT_INC**	Timer in the Interrupt Handler *(inc by every 20 ms)*
0B1F	**2847**	**1**	**IRQ_STAT**	Enabling IRQ for the expansion cards
0B20	**2848**	**1**	**INT_FLAG**	Interrupt handling in progress
0B21	**2849**	**1**	**WARM_FLAG**	**FF**: “warm” reset in progress - stop program, screen reset
0B22	**2850**	**1**	**COLD_FLAG**	**FF**: “warm” reset is disabled
0B23	**2851**	**20**	**INFUNC**	The initial part of the system functions
0B37	**2871**	**10**	**OUTFUNC**	The ending part of the system functions
0B41	**2881**	**8**	**INT_EXIT**	The initial part of the Interrupt Handler
0B49	**2889**	**2**	**STACKT**	Temporary storage for Stack Pointer

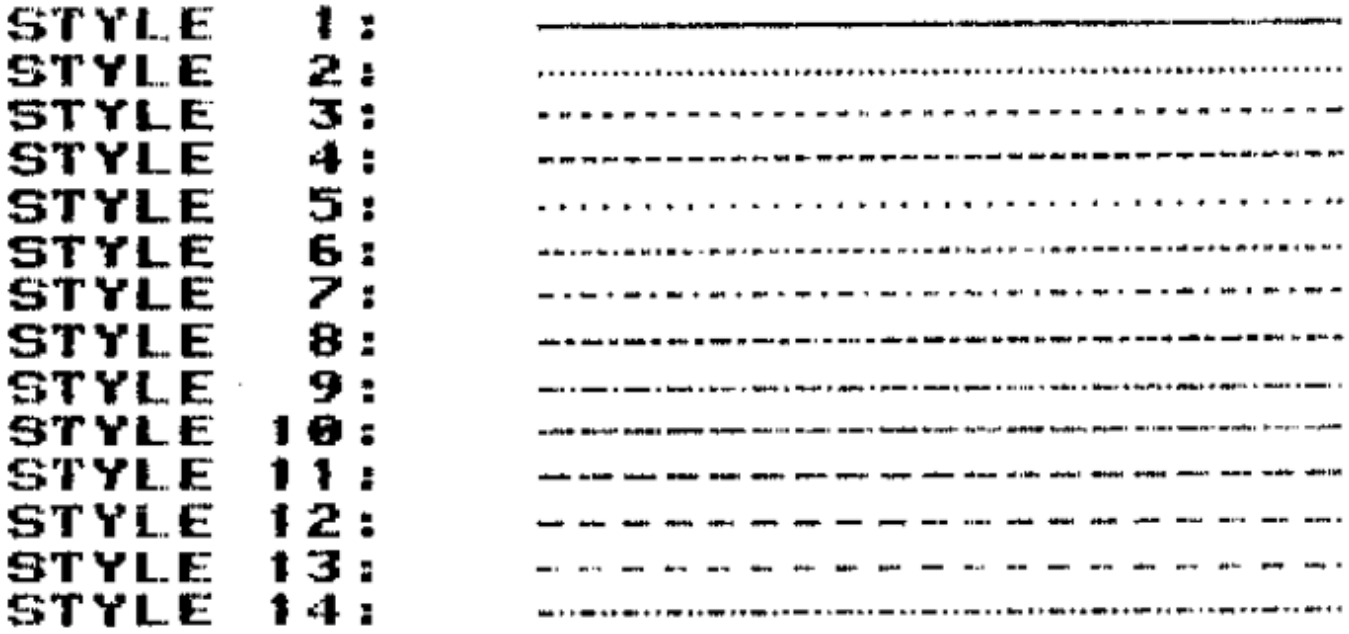
TAPE variables

hex addr.	dec.addr.	Length	Name	Decription
0B6B	**2923**	**1**	**BUFFER**	**0**: not buffered; **FF**: bufferes file
0B6C	**2924**	**1**	**REMRED**	Select tape motor control
0B6D	**2925**	**1**	**PROTECT**	**0**: not protected; **FF**: protected file
0B6E	**2926**	**1**	**EOF**	**FF**: end of file
0B6F	**2927**	**2**	**MUDDLE**	CRC base value

LINE, BORDER and TEXT variables

hex addr.	dec.addr.	Length	Name	Decription
0B4B	**2891**	**1**	**L_MODE**	Line overwriting mode *Values: **0**: overwrite; **1**: OR; **2**: AND; 3: XOR*
0B4C	**2892**	**1**	**L_STYLE**	Line drawing style
0B4D	**2893**	**1**	**INK**	Line / text color *(Palette index or color in Graphics 16)*
0B4E	**2894**	**1**	**PAPER**	Text background color *(Palette index or color in Graphics 16)*
0B4F	**2895**	**1**	**BORDER**	Border color *(Bit **7**: Intensity; **5**: Green; **3**: Red; **1**: Blue)*
0B50	**2896**	**1**	**V_FLAG**	Char overwriting flag → ***0**: overwrite; **1**: invisible ink;* ***2**: transparent background; **3**: invisible + transparent*

L_STYLE – line drawing styles



BASIC variables

hex addr.	dec.addr.	Length	Name	Decription
0008	**8**	**25**		BASIC error handling routines
0021	**33**	**14**	**USRTAB**	BASIC **EXT** instructions address table

SERIAL LINE variables

hex addr.	dec.addr.	Length	Name	Decription
0B69	**2821**	**1**	**BAUD**	Serial line speed in BAUD
0B6A	**2822**	**1**	**FORMAT**	*USART* mode
0B71	**2829**	**1**	**SER_OK**	Clock frequency **0**: OK; **FF**: not OK

KEYBOARD variables

hex addr.	dec.addr.	Length	Name	Decription
0B51	**2897**	**10**	**PICTURE**	Keyboard matrix 10 lines *(filled by the Interrupt Handler)*
0B5B	**2907**	**10**	**OLD_PICT**	Previous keyboard matrix
0B65	**2917**	**1**	**DELAY_KEY**	Auto-repeat delay
0B66	**2918**	**1**	**LOCK_KEY**	CAPS / SHIFT / ALT states
0B67	**2919**	**1**	**RATE_KEY**	Auto-repeat speed in 20 ms
0B68	**2920**	**1**	**HOLD_DIS**	**FF**: CTLR+P has no effect

Keyboard Matrix

LINE	PICTURE hex.	PICTURE dec.	B7	B6	B5	B4	B3	B2	B1	B0
0	0B51	2897	! 4	' 1	↑	/ 6	& 0	" 2	+ 3	% 5
1	0B52	2898	= 7	Ö	Ó	# *	Ü	) 9	(8	~ ^
2	0B53	2899	R	Q	@	Z	$;	W	E	T
3	0B54	2900	U	P	Ú	{ [	Ö	O	I	}]
4	0B55	2901	F	A	> <	H	\	S	D	G
5	0B56	2902	J	É	Ü	RET	A	L	K	DEL
6	0B57	2903	V	Y	LOCK	N	SHIFT	X	C	B
7	0B58	2904	M	_	SPACE	CTRL	ESC	: .	? ,	ALT
8	0B59	2905		LEFT	RIGHT	ACC	FIRE	DOWN	UP	INS
9	0B5A	2906		LEFT	RIGHT	ACC	FIRE	DOWN	UP	

Line 8: internal and Joystick #1; Line 9: Joystick #2

Get joystick example:

```
ld      A,($0B59)       ; register A = keyboard matrix line #8 – joystick #1
and     64              ; clear all bits except joystick LEFT
call    nz,MOVE_LEFT    ; if not zero then call MOVE_LEFT subroutine
ld      A,($0B59)       ; A = keyboard matrix line #8 – joystick #1 (again)
and     32              ; clear all bits except joystick RIGHT
call    nz,MOVE_RIGHT   ; if not zero then call MOVE_RIGHT subroutine
ld      A,($0B59)       ; A = keyboard matrix line #8 – joystick #1 (again)
and     8               ; clear all bits except joystick FIRE
call    nz,FIRE         ; if not zero then call FIRE subroutine
```

SYSTEM AREAS

SYSTEM areas on U0 (Page 0) memory *(length in bytes)*

hex addr.	dec.addr.	Length	Name	Decription
0100	**256**	**1600**		ASCII screen (**64** column x **25** lines) *
0740	**1856**	**960**		Matrices of definable chars (chr: **128 - 223** x **10** bytes)**
0B72	**2930**	**15**		Video working area
0B81	**2945**	**100**		**I/O** working area
0BE5	**3045**	**10**		Keyboard working area
0BEF	**3055**	**1**		Sound working area
0BF0	**3056**	**600**		Tape working area ***
0E48	**3656**	**80**		Editor working area *
0E98	**3736**	**20**	**PROG_ID**	“TV COMPUTER BASIC” text *(from ROM v1.3)*
0EAC	**3756**	**2048**	**STACK**	System STACK
16AC	**5804**	**835**		BASIC working area ****

* You are free to use these memory areas.

** You can use this memory area freely if you are not using the #128 - #223 characters.

*** If you don't save / load files, you can use this memory area freely.

**** If you are not using BASIC / SYSTEM functions, you are free to use these memory areas.