File: FutureOS.DOC (63 pages) date: 2024, January 7

The FutureOS - handbook, philosophy and conception

I. The Philosophy of the Future Operating System

II. The concept of FutureOS

III. The organisation of FutureOS

IV. The Desktop

V. The Machine-Monitor

VI. Accurate explanations about specific FutureOS features

VII. Appendix

Read this manual carefully. It will help you omit errors.

I. The Philosophy of the Future Operating System

The Origin of the Future Operating System

The idea of FutureOS was born in the late 1980ies. One main target has been to create an operating system (OS) which is capable to operate all expansion hardware for the CPC. Existing expansions are often not very compatible, one example is the incompatibility between AmsDOS, V-DOS and (X)DDOS.

Further is was planned from the beginning to develop very fast low level routines. The standard-print-routine of the CPC was always slow, even in the 80ies.

Later a graphical interface was added to this concept, the Turbo-Desktop. Its function is to perform all standard tasks fast and efficient.

Further FutureOS brings some new features, which doesn't exist on the standard CPC or other computers. Examples are working with many drives and hard-disc partitions at the same time, multi-directional filecopy, programs can have a length up to four megabyte or the I/O - porting-system...

FutureOS was planned and created as Ultra operating system, specific made for the CPC.

The speed - or do you mind if it is faster

Due to the OS philosophy maximum speed was always most important. The usage of memory is secondary. From the beginning is was planned to realize FutureOS in 64 KB Pseudo-(EP)ROM. Further the OS can be expanded by up to eight 16 KB ROMs. One example is the IDE ROM.

The CPC has a Z80 CPU at 4 MHz. To be able to compete in today's time with modern computers its needed to compensate the "weakness" of the hardware, but especially the software, by using the new software FutureOS.

Additionally the specific optimization of a program can speed it up by two orders of magnitude.

Example: The program "Zeichendemo" on the OS utilities disc needs only 23 seconds to display half a million chars on the screen. Even the 32 bit CPU of the Acorn 5000 still needs 20 seconds to do the same.

The price of the speed is the loss of compatibility to old programs.

The FutureOS structures of files are 100% compatible with the standard CPC-OS. Additionally they have been strongly expanded.

FutureOS is the ideal operating system for coders that want to max out their CPC completely.

The hardware and its administration

Every piece of periphery is served by its own driver. According to this the low level routines are very efficient. FutureOS omits any kind of so called abstraction layers, which are known to make an OS just slow. And I really don't care about an FutureOS application running on an C64 or Amiga ;-)

This documentation file will also explain the direct programming of your hardware in a very efficient way, to make your programs faster.

FutureOS provides comfortable system calls to manage up to 4 MB of expansion RAM. Programs can reach a similar size in one piece.

The price, or why is it inexpensive

FutureOS was never a commercial project. Its deeper meaning consists in establishing new possibilities to the CPCs and its users.
Now you can use your CPC forever. Just use better software and keep your well running hardware equipment.

The personal user identification number

The personalized version has several advantages: the personal user identification number(ID) of the user helps to adapt your system in the way you want. The user ID allows to log on in networks and to receive personal messages. A variety of applications uses the user ID to adapt itself to the needs of the user. Another example is the disc magazine FutureView; it displays a picture of the user and displays a personal message when loading. The probability of user ID abuse is small, since the user ID itself is anchored directly in the FutureOS ROMs.

II. The concept of FutureOS

In common

Logically the OS can be divided in three horizontal levels. At the bottom there are the low-level functions. They are used for a direct control of the hardware. The high-level functions are located in the middle. They use the low-level functions to manage more complex functions. The Desktop is located abouve everything else. It rules the system and it's needed to to hold contact with the user.

Logically you can divide the system in three levels, but in reality the system is constructed by using a pragmatic approach. To achieve maximum efficiency = performance it's often needed to ignore the three levels idea of the system.

Low-level OS functions

They manage the basal features of the system like key-scanning, print a char or string on screen or printer, memory-management, copy or fill memory very fast - faster than LDIR, disc and hard-disc management, manage external hardware, show icons and so on.

High-level OS functions

The high-level OS functions are manifold. Some are close to the hardware; some others are highly complex and go new ways. Some examples are: memory-dump, dynamic expansion RAM (=E-RAM) management (up to 4 MB), generate load- and save-tables, write to and read from disc or hard-disc, format discs, manage DIRectories etc.

Control through the Desktop

The Turbo-Desktop is the interface between artificial and biological intelligence (CPC and human being). The Turbo-Desktop has a broad variety of functions:

- Run programs - Print files or directories
- Load files - Save files
- View text and pictures - View file-headers
- Rename files - Erase files
- Copy discs (Data, System, IBM) - Copy files (multi directional)
- Format discs (Data, System, IBM) - Set time & date, alarm-time
- Call the machine-monitor - and much more ...

Further the Desktop provides the possibility to support a program directly. Every program can call the Desktop analog to an subroutine.

Then all needed functions can be accomplished by using the Turbo-Desk functions. Subsequently the user returns to the application by using the OK icon. Thus programs become more similar and easier to handle.

Grouping the OS in Desktop, Monitor and Environment

The OS can be divided vertically too. The Desktop, the Monitor and the Environment form the three columns of FutureOS. While the Desktop allows the management of data and files, the machine monitor provides direct access to the hardware.

Aside the Desktop and the machine monitor, the Environment may be the most important column. It allows the coder to generate very efficient programs. An OS library provides additional support.

III. The organisation of the Operating System

Memory-management

First some information about the first 64 KB and the therein included system-variables. Afterwards some words about the expansion RAM. Please take also a look at the files **#OS-VAR.ENG** and **#EQU-API.ENG**!
The OS itself is divided in four 16 KB ROMs, named A, B, C and D. They can have any physical ROM-numbers (ROM-select) they want. Only ROM A need a number between 0 and 15. The ROM-select of the OS ROMs is fixed, but can be adapted using a program. Only one of the ROMs can be selected between &C000 and &FFFF at one time.
The E-RAM is being managed in 16 KB blocks, which can be banked switched in between &4000 and &7FFF.

Memory Map of the Future Operating System

The standard RAM (the first 64 KB) can be divided into four 16 KB blocks (0, 1, 2 and 3).

```
Block 0: from &0000 to &3FFF
   Read: 16K RAM or lower-ROM. Charset must be between &3800 and &3FFF
  Write: 16K RAM (RST vectors when using Interrupt Mode 1)
```

You shouldn't store data in this block for longer times, because the Desktop uses it sometimes as buffer. A program can freely use this block. But think about the charset between &3800 and &3FFF.

```
Block 1: from &4000 to &7FFF
   Read: 16K RAM / Standard-RAM or expansion-RAM (0-31 - 255)
  Write: 16K RAM / or Memory Mapped I/O (CPC Plus)
```

Normally the standard RAM is selected (configuration &7FC0), but there can be one of 32 or rather 255 E-RAMs be switched in, if connected (configuration: &C4,&C5,..,&FF / 4 MB: Highbyte &78XX-&7FXX).

```
Block 2: from &8000 to &BFFF
   Read: 16K RAM the only COMMON Block
  Write: 16K RAM the only COMMON Block
```

This 16 KB RAM is always switched on under FutureOS. It contains the system variables. The OS uses maximal 8 KB (between &A000 and &BFFF).

```
Block 3: from &C000 to &FFFF
   Read: 16K RAM or an Expansion-ROM, normally a FutureOS ROM
  Write: 16K Video-RAM or ROM-RAM-Box (or similar construct)
```

The entry points of the FutureOS API is located in this block, furthermore you can call system functions directly. Any data written in this block will write it in the video RAM.

The system RAM

Now the system-variables are described, the memory between &A000 and &BFFF is used in the following way, shown downwards:

&BE02 to &BFFF:	510 bytes stack => 255 elements (16 bit)
&BD00 to &BE00:	(&BE00 inclusive) 257 bytes INTERRUPT MODE 2 vectors
&BC80 to &BCFF:	128 byte file-header (Background-program)
&BC00 to &BC7F:	128 byte file-header (Foreground-program)
&B800 to &BBFF:	1024 bytes as 1 KB memory for system variables (Further information: Look below)
&B000 to &B7FF:	2048 bytes text-screen buffer. This 2 KB block can be freely used by the user or a program.
&A000 to &AFFF:	This area contains the file-tagging-bytes (FTB). They give information about all reachable files: (not) tagged, read-only-, system and archive-flag.

If the DIRectory of a drive is not read, the user or a program can use the corresponding area. The system-variables TURBO_A up to TURBO_M tell if a drive is active, if the DIR of a drive has been read and if the corresponding RAM can be used.

A000-A0FF:	256 tagging bytes for drive A, FDC 0 (internal, on PCB)
A100-A1FF:	256 tagging bytes for drive B, FDC 0 (internal)
A200-A2FF:	256 tagging bytes for drive C, FDC 0 (internal)
A300-A3FF:	256 tagging bytes for drive D, FDC 0 (internal)
A400-A4FF:	256 tagging bytes for drive E, FDC 1 (external, Vortex)
A500-A5FF:	256 tagging bytes for drive F, FDC 1 (external, F1-D)
A600-A6FF:	256 tagging bytes for drive G, FDC 1 (external, or)
A700-A7FF:	256 tagging bytes for drive H, FDC 1 (external, M1-D)
A800-A9FF:	512 tagging bytes for hard-disc partition I (Dobbertin)
AA00-ABFF:	512 tagging bytes for hard-disc partition J (HD20)
AC00-ADFF:	512 tagging bytes for hard-disc partition K
AE00-AFFF:	512 tagging bytes for hard-disc partition L or M(em) drive

Every file-tagging-byte (FTB) has the following structure:

Bit 0 - Zero => not tagged /// One => file is tagged
Bit 1 - Zero => not tagged before /// One => was tagged before
Bit 2 - reserved
Bit 3 - reserved
Bit 4 - reserved
Bit 5 - RW / RO ===> structure like in DIR (32er original)
Bit 6 - DIR / SYS ===> structure like in DIR (32er original)
Bit 7 - NoARC / ARC ===> structure like in DIR (32er original)

Under the Turbo-Desktop a tagged file is shown underlined. A file that was tagged before, but has already been handled is displayd in striked out letters.
The file attributes RO, SYS, ARC are displayed by inverting the corresponding letter of the file name extension (last three letters of the 11 letters of the file name).

Explanation of the system-RAM-variables

In the following a short description is given about every RAM-variable. Please be careful with some of them, because a wrong value can influence the system stability.

ALL these RAM variables are collected in the file **#EQU-API.ENG** and can be accessed by the programmer.

The string **EQU** means **"="**, both values beside are EQUal. The expression at the left side is equal to the expression at the right side.

DS stands for "define space". The value afterwards gives information about how much space the corresponding system-variable uses.

Abbreviations:

DR	=	Floppy disc DRive
DIR	=	DIRectory (floppy disc or hard-disc partition)

You can connect two FDC765 (floppy-disc-controller) to your CPC. The internal controller is the one of Amstrad (or Dobbertin). The external controller is from Vortex. They use different I/O addresses, but are used in the same way. The two following variables show the MSR (main status register) of each FDC765:

```
FDC0_ST EQU &FB7E   ;main-status-register(MSR), internal FDC
FDC1_ST EQU &FBF6   ;MSR, external FDC(Vortex)
```

The variables itself

The above variables show at which address the file-tagging-bytes of a disc or hard-disc starts in RAM (look at the following side, too).

```
TMS_A  EQU   &A000 ==>    drive A      / <T>agging <M>ass <S>pace_<drive-A>.
TMS_B  EQU   &A100 ==>    drive B    /   (internal FDC, on 6128 PCB)
TMS_C  EQU   &A200 ==>    drive C   /
TMS_D  EQU   &A300 ==>    drive D /
TMS_E  EQU   &A400 ==>    drive E      / ...first Vortex drive, called E
TMS_F  EQU   &A500 ==>    drive F    /   ...second Vortex drive, called F
TMS_G EQU    &A600 ==>    drive G   /       External FDC from Vortex, used in
TMS_H EQU    &A700 ==>    drive H /         F1-D and M1-D drives.
TMS_I  EQU   &A800 ==>    Dobbertin hard-disc HD20, first partition I
TMS_J  EQU   &AA00 ==>    second partition, called J
TMS_K  EQU   &AC00 ==>    third partition, called K
TMS_L  EQU   &AE00 ==>    fourth partition, called L
```

TAR16 EQU &1800 ==> That 8 KB buffer (&1800-&37FF) is used to convert the 32ers DIR (32 bytes per entry) to the 16ers DIR (16 bytes) due to routine DIRWA.

- The 32ers DIR equals the DIR on (hard)disc.
- The 16ers DIR is shown on the screen.

TXT_SCR EQU &B000 ==> Start of the 2 KB text-screen-buffer (for DUMP)

Now the variables between B800 and BBFF are declared. These variables are sorted by sense (for RAM-address look at file **#EQU-API.ENG**).

TAS_S1 DS 64 ==> 64 (32*2) bytes for Mode 1 terminal JP table.
TAS_S2 DS 64 ==> 64 (32*2) bytes for Mode 2 terminal JP table.
Look at chapter "print chars" for more information.

RAMCHAR DS 1 ==> Defines the screen mode 0...3 (bits 1, 0) and defines if the ROM charset (bit 2 = 0) or the RAM charset (bit 2 = 1) is used. The charset is always located between &3800 and &3FFF.

C_POS DS 2 ==> These two bytes contain the actual Cursor-POSiton. To calculate the actual V-RAM address you have to add value 1 in Mode 2, or you must add value 1 in Mode 2.

- Mode 1: &BFFE to &C7FE.
- Mode 2: &BFFF to &C7FF.

FDC_RES DS 7 ==> The seven bytes of the last result-phase of the FDC 0 (internal, Amstrad) or 1 (external, Vortex) are stored here. They provide information about the success of the last disc-operation.

MO_ST DS 1 ==> this byte allows the manipulation of the drive-motor-status of all floppy disc drives. Normally MO_ST is set to &00, therefore all drive-motors are off when no drive-access is made. But if MO_ST contains the Byte &FF all drive-motors are running permanently. Therefore you don't need the HEAD-LOAD-TIME, in the case of FutureOS the READY-TIME. How to switch the drives on or off?

Drive-motors on: `LD BC,&FA7E : LD A,&FF : OUT (C),A ;MSB = 1`
Drive-motors off: `LD BC,&FA7E : OUT (C),C ;Bit 0=0 => MSB = 0`

The following two variables contain the number of read/write tries:
FDCLSA DS 1 ==> actual try to read/write disc FDC 0/1 (variable)
FDCLSV DS 1 ==> number of tries to read/write disc FDC 0/1 (constant)

Every floppy disc drive has its own specific step-rate-time. The step-rate-times of all eight drives A...H are located in RAM at DSWZ. Every drive has its own one byte variable, the lower nibble MUST be zero. For example:

```
 Floppy:  A ! B ! C ! D ! E ! F ! G ! H ! The step-rate-time variables
------------+---+---+---+---+---+---+---+ for all eight floppy-disc-
 DSWZ DB &A0,&A0,&E0,&E0,&E0,&E0,&E0,&E0! drives A...H.
```

AKT_ROM DS 2 ==> The low-byte contains the number of the at present selected ROM (&00-&FF). It's followed by the high-byte &DF. This allows the following construction:

```
LD  BC,(AKT_ROM)    ;B = &DF, C = actual ROM select...
OUT (C),C           ;switch in on (again)
```

AKT_RAM DS 2 ==> The low-byte selects the actual RAM. If is is &C0 then the first 64 KB are switched in. In contrast the values &C4 - &FF select an Exp.-RAM, that is banked in between &4000 and &7FFF. The following high-byte &7F makes this code possible:

```
LD  BC,(AKT_RAM)    ;load RAM configuration in BC
OUT (C),C           ;switch RAM on
```

If the CPC has more than 512 KB Expansion-RAM, then the high-byte can contain &78 - &7E instead of &7F.

TAST_R0 DS 10 ==> This 10 bytes contain the 10 lines (0 to 9) of the PSG key-matrix. That makes 10 * 8 bit = 80 bit. If one of the 80 keys of the CPC is pressed, its corresponding bit is cleared to zero. If a key is not pressed, its bit is set to 1.
Use routine HOLETST to fill the bytes of this matrix with data, since the key matrix is not scanned by any interrupt (to save CPU time).

There are 13 times 8 bytes, they are used to manage different devices.

Every storage device (Floppy, HD, MM) is using eight bytes:

TURBO_A	DS 8	==>	Turbo Desk drive A (Amstrad controller, internal FDC)
TURBO_B	DS 8	==>	Turbo Desk drive B
TURBO_C	DS 8	==>	Turbo Desk drive C
TURBO_D	DS 8	==>	Turbo Desk drive D
TURBO_E	DS 8	==>	Turbo Desk drive E (Vortex controller, external FDC)
TURBO_F	DS 8	==>	Turbo Desk drive F
TURBO_G	DS 8	==>	Turbo Desk drive G
TURBO_H	DS 8	==>	Turbo Desk drive H
TURBO_I	DS 8	==>	Turbo Desk HD partition I (Dobbertin hard-disc HD20)
TURBO_J	DS 8	==>	Turbo Desk HD partition J
TURBO_K	DS 8	==>	Turbo Desk HD partition K
TURBO_L	DS 8	==>	Turbo Desk HD partition L
TURBO_M	DS 8	==>	Turbo Desk virtual RAM-drive M (MM)

These groups of 8 bytes give the following information. The sense is the same for all drives or partitions.

*** Byte 0** - drive tagging byte. It shows if the drive is active ect.
The bits 7, 6, 5 and 4 contains the format-number of the actual disc:

&0X ==> HD partition OR no drive access until now OR unknown format.
&1X ==> the actual disc has VORTEX format.
&2X ==> IBM ---------- format.
&4X ==> SYSTEM ----- format.
&8X ==> FutureOS --- format.
&CX ==> DATA -------- format.

Bit 3 shows if the DIR(ectory) had been manipulatet, like after the renaming or erasing of some files. This bit is only relevant if bit 0 is set to 1.
Bit 3 = 0 ==> DIR wasn't manipulated until now.
Bit 3 = 1 ==> DIR is manipulated because of a file operation.

Bit 2 contains the head-number. It is only relevant when you use a single-side format. If the disc is Vortex formatted, it should be = 0 in every case.
Bit 2 = 0 ==> head-number 0 is used (standart)
Bit 2 = 1 ==> head-number 1 is used (format dependent)

Bit 1 gives information if the drive is connected and active. If a drive is not connected or not set active, the corresponding drive-icon is show with bars (in the Desktop).
Bit 1 = 0 ==> the drive is connected and ready to use.
Bit 1 = 1 ==> the drive is not connected or inactive.

Bit 0 shows if the drive is tagged or not. You can only work with tagged drives. If a drive is not tagged it is ignored.
Bit 0 = 0 ==> the drive is not tagged. All other bits can be ignored.
Bit 0 = 1 ==> the drive is tagged, you can work with it.

Now the other 7 bytes are declared. Their values are only important if a drive is set active / is tagged (Bit 0 of Byte 0 is set).

Byte 1 - RAM block &C4-&FF or &C0	(32er DIR, like on disc)
Byte 2 - Start-address (in Pages, each 256 bytes) &40…&7F	(32er DIR)
Byte 3 - Length or DIR in pages &00...&40	(32er DIR)

Byte 4 - RAM block &C4-&FF or &C0	(16er DIR, is shown on the screen)
Byte 5 - Start-address (in pages) &40..&7F	(16er DIR)
Byte 6 - Lenght (in pages) &00..&20	(16er DIR)

Byte 7 - number of 1 KB entry-pages, every page contain 64 entrys.

The 32er DIR equals the directory on the disc. It is RAM buffered.
The 16er DIR is a text-version of the 32er DIR. It is created to show the files on the screen (under Turbo Desk).

TURBO_X DS 2 ==> The low-byte contains the number of the highest free external RAM block &C4, ..., &FF, all E-RAMs above are used for RAM-buffered DIRs. If it contains &C0, then the E-RAM is used up.

The high-byte contains the highest free RAM address in the above named E-RAM, diveded by &100: &41 - &80.

Example: The high-byte &78 stands for the RAM address &7800, the RAM below is freely usable.

One general example: TURBO_X contains &FF, &80 = &80FF if 512 KB E-RAM are connected and no DIRectory has been read.

Using the machine-monitor you can give every Z80-register a value to call an mc-routine subsequently. You can use these bytes freely.

REG_AF1 DS 2 ==> register AF ==> first register set
REG_BC1 DS 2 ==> register BC
REG_DE1 DS 2 ==> register DE
REG_HL1 DS 2 ==> register HL

REG_AF2 DS 2 ==> register AF' ==> second register set
REG_BC2 DS 2 ==> register BC'
REG_DE2 DS 2 ==> register DE'
REG_HL2 DS 2 ==> register HL'

REG_IX DS 2 ==> register IX ==> index registers
REG_IY DS 2 ==> register IY

REG_SP DS 2 ==> register SP ==> special registers
REG_R DS 1 ==> register R
REG_I DS 1 ==> register I ==> Attention: It MUST be &BD !!
REG_PC DS 2 ==> register PC

The following variables can be used freely by the user. They're called the user-registers and are used to give values to certain routines.
But use the REG??_? variables only to store values for a short time, because there are much routines which use these registers too.

REG08_0 DS 1 ==> USER-register 0 (8 Bit)
REG08_1 DS 1 ==> USER-register 1 (8 Bit)
REG08_2 DS 1 ==> USER-register 2 (8 Bit)
REG08_3 DS 1 ==> USER-register 3 (8 Bit)
REG08_4 DS 1 ==> USER-register 4 (8 Bit)
REG08_5 DS 1 ==> USER-register 5 (8 Bit)
REG08_6 DS 1 ==> USER-register 6 (8 Bit)
REG08_7 DS 1 ==> USER-register 7 (8 Bit)

REG16_0 DS 2 ==> USER-register 0 (16 Bit)
REG16_1 DS 2 ==> USER-register 1 (16 Bit)
REG16_2 DS 2 ==> USER-register 2 (16 Bit)
REG16_3 DS 2 ==> USER-register 3 (16 Bit)
REG16_4 DS 2 ==> USER-register 4 (16 Bit)
REG16_5 DS 2 ==> USER-register 5 (16 Bit)
REG16_6 DS 2 ==> USER-register 6 (16 Bit)
REG16_7 DS 2 ==> USER-register 7 (16 Bit)
REG16_8 DS 2 ==> USER-register 8 (16 Bit)
REG16_9 DS 2 ==> USER-register 9 (16 Bit)

REG32_0 DS 4 ==> USER-register 0 (32 Bit)
REG32_1 DS 4 ==> USER-register 1 (32 Bit)

The OS must know how much files are on a disc. The following variables contain the number of different files per disc. Dependent on the format that can be 0 up to 64 (3") or 128 (Vortex) or 512 (hard-disc).

TMD_A DS 2 ==> number of files on drive <A> (internal)
TMD_B DS 2 ==> ... <B>
TMD_C DS 2
TMD_D DS 2

TMD_E DS 2 ==> external FDC (Vortex F1-S, F1-D or Dobbertin D-DOS)
TMD_F DS 2 ==> second drive, under the OS called F
TMD_G DS 2
TMD_H DS 2

TMD_I DS 2 ==> 20 MB Dobbertin hard-disc, first partition, called I
TMD_J DS 2
TMD_K DS 2
TMD_L DS 2

TMD_M DS 2 ==> Memory Floppy (like Otten & Fecht), inactive up now.

The following 8 bytes contain the data of the real-time-clock. The ROM-version of the SETUP contain the burning time & date of the Eprom.

UHR_00 DS 1 ==> byte 0: hundredth and tenth of second
UHR_SEK DS 1 ==> byte 1: second
UHR_MIN DS 1 ==> byte 2: minute
UHR_STU DS 1 ==> byte 3: hour
UHR_WOT DS 1 ==> byte 4: day of week. Clear bit 5 to start RTC
UHR_TAG DS 1 ==> byte 5: day of month
UHR_MON DS 1 ==> byte 6: month of year
UHR_JAR DS 1 ==> byte 7: year (Decade and Year, f.e. 9 = 2009)

UHR_ROM DS 2 ==> The low-byte provides the ROM number of the Dobbertin dxs, or M4 Real-Time-Clock (RTC). It's the number of the RTC itself or the M4 ROM. The high-byte must be &DF, to allow the following code:

```
LD  BC,(UHR_ROM)    ;B = &DF, C = ROM numb. of Dobb. RTC
OUT (C),C           ;switch smart-watch on (as ROM)
```

The two following variables contain the actual size of the screen. In general the values for Mode 2 are given, that doesn't depend on the real screen mode.

It's not allowed to go over these maximal-values when positioning the cursor, else errors could happen. In the worst case bytes can be written anywhere in the RAM. Therefore, if using control-codes in strings, don't use bigger values.

MAX_CRX DS 1 ==> maximum columns per line, mode dependent (max. 104!)
MAX_CRY DS 1 ==> maximum lines per screen, mode dependent (max. 40!)

MAX_RAM DS 1 ==> This value contains the number of 16K expansion-RAM blocks which can be used freely without a memory hole. Start at RAM &C4. But look in XRAM_?? if you're allowed to use them!

ADROM_X DS 1 ==> 8 bit number (ROM select) of an FutureOS expansion ROM. # of E-ROM which used OS functions at last.

ADROM_A DS 2 ==> Number of FutureOS ROM A, followed by the byte &DF. This allows:
`LD BC,(ADROM_A):OUT (C),C` to switch in the FutureOS ROM A.

FREEZE DS 1 ==> The upmost bit (bit 7) is used for the USB token of the USB mouse of the Albireo expansion. Bits 6 to 0 are unused.

DIRIN DS 1 ==> gives information if any DIR has been read into RAM. If a DIR is read, this variable contains the drive number A..M = &00-&0C.
If DIRIN is &FF, no DIRectory has been read. In this case TURBO_A...M don't matter.

The **12 config / setup bytes** provide information about a variety of OS features. More details are provided in the file '**#OS-VAR.ENG**'

KF_MED DS 1 ==> Tells if the SPARtan mode is activated, if numbers are entered in hexadecimal or decimal system. Tells if a CPC Digiblaster, PlayCity, MultiPlay, Albireo, M4 or VN96 network adapter is present.

KF_FDHD DS 1 ==> tells if an internal or an external FDC is connected, if a Dobbertin HD20 or Vortex Winchester is connected. And if real-time-clocks (Dobbertin, dxs, Happy), a tape deck and the CPC Booster are available.

KF_AB DS 1 ==> contains the number of tracks and sides of drive A and drive B, how much inches they have and if they're useable.

KF_CD DS 1 ==> "" ... of drive C and drive D.

KF_EF DS 1 ==> contains the number of tracks and sides of drive E and drive F, how much inches they have and if they're useable.

KF_GH DS 1 ==> "" ... of drive G and drive H.

KF_MEM DS 1 ==> Tells which blocks of 512 KB of the total maximum of adressable expansion RAM of 4 MB are connected.

KF_SIO DS 1 ==> information about connected serial interface(s), if the printer port has 7 or 8 bits, if a monochrome or a color screen is connected.

KF_CPC DS 1 ==> this byte encodes the used CPC-type and the language of the user. A program can look there and decide how to run (selection of language). Tells if an X-MASS or SYMBiFACE II/III is present.

KF_VERS DS 1 ==> Defines the version of FutureOS (personal or generic), format of time and date (icons or numbers), auto-DIR and if a speech synthesiser is present (LambdaSpeak, SSA-1 or dk'tronics).

KF_OSUN DS 2 ==> Contains the personal serial, user-number of the user.

The following five variables/bytes are used by the routines SC_AB, SC_AU and SCRI. They are used by the Desktop to browse through DIRs.

TDRAM	DS 1	==>	the RAM block of the 16er DIR of the actual drive.
TDHST	DS 1	==>	high-byte of the start-address of the 16er DIR.
TDANZ	DS 1	==>	number of 1 KB pages.
TDAKT	DS 1	==>	actual 1K page (shown on the Desktop-screen).
TDLWK	DS 1	==>	actual drive.

FutureOS provides four keyboard levels. This four times 80 bytes determine the value (&00-&FE) of a pressed key:

TAST_N	DS 80	==>	80 bytes for keymatrix 10 * 8 ==> NORMAL level
TAST_S	DS 80	==>	80 bytes for keymatrix 10 * 8 ==> SHIFT level
TAST_C	DS 80	==>	80 bytes for keymatrix 10 * 8 ==> CONTROL level
TAST_SC	DS 80	==>	80 bytes for keymatrix 10 * 8 ==> SHIFT/CONTROL lev.

The first 512 KB expansion RAM

You can connect up to 4 MB expansion RAM to the CPC; the first 512 KB bear a special meaning under FutureOS: This RAM is managed in blocks of 16 KB. Single blocks are banked in between &4000 and &7FFF.

Every one of the 32 blocks of 16 KB has its own 1 byte variable, which provides information if this block is connected and in which way it is being used. Such a variable is in the following way constructed:

Bit 7	==>	= 1 ==> RAM is occupied through a loaded file.
Bit 6	==>	= 1 ==> RAM is used through a multitasking-routine/program.
Bit 5	==>	= 1 ==> ??? reserved!!!
Bit 4	==>	= 1 ==> RAM is used by the user, the OS ignores it!
Bit 3	==>	= 1 ==> RAM is used as short-time-buffer.
Bit 2	==>	= 1 ==> RAM is used as long-time-buffer.
Bit 1	==>	= 1 ==> RAM buffers one or more DIRectories (disc or HD).
Bit 0	==>	= 1 ==> RAM is connected /// = 0 ==> RAM doesn't exist.

Only one of the Bits 1-7 is allowed to be set to "1" at the same time.
These bits inform in which way the 16 KB RAM block is used.

These are names of the 32 E-RAM variables:

```
XRAM_C4, XRAM_C5, XRAM_C6, XRAM_C7, XRAM_CC, XRAM_CD, XRAM_CE, XRAM_CF
XRAM_D4, XRAM_D5, XRAM_D6, XRAM_D7, XRAM_DC, XRAM_DD, XRAM_DE, XRAM_DF
XRAM_E4, XRAM_E5, XRAM_E6, XRAM_E7, XRAM_EC, XRAM_ED, XRAM_EE, XRAM_EF
XRAM_F4, XRAM_F5, XRAM_F6, XRAM_F7, XRAM_FC, XRAM_FD, XRAM_FE, XRAM_FF
```

End of each variable XRAM_C4... _FF corresponds to the low byte of its physical E-RAM select using port &7Fxx.

More than 512 KB E-RAM

The expansion RAM of the CPC can be divided into blocks of 512 KB. Every single block can be accessed by using one of the following I/O port addresses: &7Fxx (first 512 KB E-RAM), &7Exx, &7Dxx, &7Cxx, &7Bxx, &7Axx, &79xx or &78xx.

Thereby the usage of the low byte is similar for every one of the 512 KB blocks. Like it is done for the first 512 KB E-RAM (&7Fxx). This way you can comfortably manage up to 4 MB E-RAM.

If the CPC has more than 512 KB expansion RAM connected, then this additional expansion RAM is managed by using the following system variables.

FutureOS initializes this variables at the start of the system.

X4RXE DS 1 ==> This variable shows if there are more than 512 KB E-RAM connected or not.
If bit 7 = 0 then the CPC has max. 512 KB E-RAM.
If bit 7 = 1 then there is more than 512 KB E-RAM connected.

Bit 4 has a special function: If this bit is set then exactly 64 KB E-RAM can be accessed using values &C4-&C7 at port &7ECx. This is the case when using an X-MEM. The internal E-RAM of the CPC6128 and 6128 Plus can be used via port &7ECx this way: Take togerther 640 KB E-RAM.

The bits 3, 2, 1 and 0 of variable X4RXE show if E-RAM using the ports &7EFx, &7EEx, &7EDx and &7ECx E-RAM can be accessed. If one of this bits is cleared to 0, then the E-RAM is NOT connected. But if a bit is set to 1 then the corresponding 128 KB E-RAM block can be accessed.

If the variable X4RXE contains the value zero, then the CPC has not more than 512 KB E-RAM connected.

The following table of variables shows which E-RAM blocks are connected above the first 512 KB E-RAM block. Every bit of the four variables symbolizes a block of 128 KB E-RAM. This block can be accessed by using the I/O port &7NNx. Bit 4 of X4RXE equals 64 KB.

Var	Bit	7	6	5	4	3	2	1	0
X4RXE	512K+?	-	-	&7EC4-7	&7EFx	&7EEx	&7EDx	&7ECx	
X4RDC		&7CFx	&7CEx	&7CDx	&7CCx	&7DFx	&7DEx	&7DDx	&7DCx
X4RBA		&7AFx	&7AEx	&7ADx	&7ACx	&7BFx	&7BEx	&7BDx	&7BCx
X4R98		&78Fx	&78Ex	&78Dx	&78Cx	&79Fx	&79Ex	&79Dx	&79Cx

Now a cleared bit tells that the corresponding block of 128 KB E-RAM is not connected.
But if the bit is set then the corresponding block of 128 KB E-RAM is usable.
You can find further information in the file with the name **#OS-VAR.ENG**.

The described four variables divide the 4 MB E-RAM logical into blocks of 128 KB size. But FutureOS manages the E-RAM in blocks of 16 KB.

A single 16 KB block can be banked in between &4000 and &7FFF.

Beginning at variable X4RAM the system RAM contains 28 bytes. The 224 bits of this 28 bytes manage the part of the E-RAM above the first 512 KB of E-RAM. The first 512 KB of the E-RAM are not considered in this bits, because they are already managed by the XRAM variables.

Every single bit of this 224 bits is associated to one 16 KB block of E-RAM. If such a bit is cleared, then the corresponding 16 KB block is currently not in use. But if a bit is set to 1 then the block is already occupied.

The following table shows the association between bits and 16 KB E-RAM blocks:

1. Byte, 8. Bit -> Block &7EC4
1. Byte, 7. Bit -> Block &7EC5
1. Byte, 6. Bit -> Block &7EC6
.
.
.
28. Byte, 3. Bit -> Block &78FD
28. Byte, 2. Bit -> Block &78FE
28. Byte, 1. Bit -> Block &78FF

In the FutureOS machine-monitor the RAM/ROM state can be set for all operations. The following variables contain its status:

MON_ROM DS 1 ==> &82 ==> lower ROM on /// &86 lower ROM off.
MON_RAM DS 1 ==> number of the E-RAM block &C4-&FF or &C0 norm. 64 KB
MON_MMB DS 1 ==> &A0 ==> Memory mapped I/O off // &B8 ==> MM active.
Attention: Only the CPCplus has Memory Mapped I/O.

The routine XWART has its own variables. They decide about the waiting times. Look at file 'API-A-EN.DOC' for more precise information.
VZ_MOD DS 1 ==> waiting mode = 0 then first wait for key str. else not
VZ_AG DS 2 ==> waiting for first key strike (reload value)
VZ_AA DS 2 ==> waiting for first key strike (actual left time)
VZ_FG DS 2 ==> waiting for next key strike (reload value)
VZ_FA DS 2 ==> waiting for next key strike (actual left time)

DEFINT DS 1 ==> This variable defines if interrupts are allowed (EI) or not (DI). Beware! FutureOS works in Interrupt-Mode 2! NOT in IM 1.
If DEFINT contains the value &00 all interrupts are switched off (DI).
But if DEFINT is set to &FF then the interrupts are active (EI).
Interrupts are usually switched off, so the both the IM 1 and the MI 2 entry can be used by applications.

M4_ERAM DS 1 ==> Defines a block of 16 KB E-RAM inside the first 512 KB of the expansion memory (&C4 - &FF). This block is used for the management of the SD card of the M4 expansion.

IDE_XR DS 1 ==> Defines the ROM number of the XROM which gets called when clicking at the IDE- / mass storage icon (address &C009). On &00 no such XROM does exist. This variable is set by the XROM itself.

The OK icon allows jumping back from the Desktop to a program. It has some own 16 bit variables:
OK_ADR DS 2 ==> ADDRess which is called through the OS icon (&????).
OK_ATE DS 2 ==> This value added to (OK_ADR) must be &FFFF.
OK_BLK DS 2 ==> 16K RAM switched on through OK (&7F00 + &C0, &C4-&FF).
OK_BTE DS 2 ==> This value added to (OK_BLK) must be &FFFF.

CAPS DS 1 ==> defines the Caps-Lock-mode. If CAPS contains &00 every key is used like usual, but if CAPS is set to &FF only BIG characters are shown when you press a letter key.

<u>The following 16 bit variables are used by the I icon:</u>
I_ADR DS 2 ==> ADDRess which is called through the I icon (&????).
I_ATE DS 2 ==> value added to (I_ADR) must be &FFFF.
I_BLK DS 2 ==> banked RAM at I icon call &7F00 + &C0,&C4,&C5,..,&FF.
I_BTE DS 2 ==> value added to (I_BLK) must be &FFFF.

The following variables can be used to manage the printer spooler:
DRUMEM DS 1 ==> phys. block-nr. first E-RAM block Spooler &C4-&FF
DRUBAZ DS 1 ==> number of direct following E-RAM blocks 0..31 for it.

A putative RAM-disc is managed through the two following variables:
MMFMEM DS 1 ==> first E-RAM block of RAM-disc &C4-&FF
MMFBAZ DS 1 ==> number of direct following E-RAM blocks 0..31

HI_MEM DS 2 ==> shows the highest free byte, like in basic, normally it's &9FFF. But it is not used through the OS. If a program uses RAM is should set this variable correct.

AUH_00 DS 1 ==> 8 bytes, like UHR_??? data
AUH_SEK DS 1 ==> It mirrors the old time
AUH_MIN DS 1 ==> before the last read of the RTC
AUH_STU DS 1
AUH_WOT DS 1
AUH_TAG DS 1
AUH_MON DS 1
AUH_JAR DS 1

WECK_ST DS 1 ==> = &00 ==> NO alert-time // &FF ==> alert is activ
WECK_ZS DS 1 ==> second alert-time
WECK_ZM DS 1 ==> minute
WECK_ZH DS 1 ==> hour

BILD_SS DS 2 ==> start-address screen-saver
BILD_RB DS 1 ==> E-RAM of the screen-saver
BILD_ST DS 1 ==> &00 ==> NO screen-saver loaded // &FF ==> saver active

BSS_WW DS 2 ==> screen-saver reload-counter &0000-&FFFF
BSS_WA DS 2 ==> actual decreasing value of screen-saver, fewer BSS_WW

HGB_RB DS 1 ==> RAM &C4-&FF for a 16 KB background picture (Desktop)
HGB_ST DS 1 ==> Background pic state, &00 => no Pic, &FF => Pic loaded

L_RAM DS 1 ==> &00 => lower RAM not buffered, else &C0-&FF => physical RAM number where the lower RAM (&0000-&3FFF) is buffered.

BORDER DS 1 ==> hardware-color value of the Border

INK_0 DS 1 ==> hardware-color INK 0 (background)
INK_1 DS 1 ==> hardware-color INK 1 (pen 1)
INK_2 DS 1 ==> hardware-color INK 2 (pen 2)
INK_3 DS 1 ==> hardware-color INK 3 (pen 3)

FDC_ERR DS 2 ==> address FDC765-error-handler
FDE_RAM DS 2 ==> error-handler-RAM &C0-&FF, followed by the byte &7F

JT_JH DS 1 ===> This variable contains the actual Millennium (in upper nibble) and the actual century (lower nibble). The variable JT_JH is BCD encoded, for example the value &20 means the years 20??. After the end of year 9999 an update of FutureOS must be performed.

F2AMS DS 1 ===> If this variable contains the value &FF, then FutureOS has been started from AMSDOS as a subroutine. In this case the first 48 KB of RAM (without the screen RAM) have been saved in the E-RAM. But if this variable is &00, then FutureOS has been started in the normal way.

HEGP2 DS 1 ===> Bit 0 tells if the Hegetron Grafpad 2 is connected to the CPC (bit 0 = 1) or not (bit 0 = 0). Bit 1 contains the status of the Grafpad. If this bit is cleared to 0 then the Grafpad is not active. But if it is set to 1 then the Grafpad is currently in use.

The following system variables are used to manage the CPC-IDE, the IDE part of the SYMBiFACE II and the IDE8255. These variables are being used by the IDE ROM. Hard discs of a maximum capacity of up to 128 Peta Byte can be managed by IDE ROM.

IDE DS 1 ==> Variable IDE shows if IDE devices are connected and which kind of IDE devices are available on the IDE expansion. This variable is also used to select the IDE adapter that shall be used for the next operation (IDE8255 or CPC-IDE / SYMBiFACE II). In addition it selects if the Master or the Slave device shall be used. Further the variable IDE tells if LBA28 or LBA48 is currently being used. If variable IDE equals &00, then NO IDE device is connected. Single bits of this variable have the following function:

Bit 7 = 0 --> IDE8255 selected
= 1 --> CPC-IDE/SF2 selected

Bit 6 = 0 --> no IDE8255
= 1 --> IDE8255 connected

Bit 5 = 0 --> no CPC-IDE/SF2
= 1 --> CPC-IDE/SF2 connected

Bit 4 = 0 --> IDE device 0 selected (see Bit 7)
= 1 --> IDE device 1 selected (see Bit 7)

Bit 3 = 0 --> CPC-IDE/SF2 device 0 works with LBA 28 (Master)
= 1 --> CPC-IDE/SF2 device 0 works with LBA 48 (Master)

Bit 2 = 0 --> CPC-IDE/SF2 device 1 works with LBA28 (Slave)
= 1 --> CPC-IDE/SF2 device 1 works with LBA48 (Slave)

Bit 1 = 0 --> IDE8255 device 0 works with LBA28 (Master)
= 1 --> IDE8255 device 0 works with LBA48 (Master)

Bit 0 = 0 --> IDE8255 device 1 works with LBA28 (Slave)
= 1 --> IDE8255 device 1 works with LBA48 (Slave)

IDE_DEV DS 1 ==> This variable tells if a certain IDE device is ready to use at the moment or not. If the variable contains the value &00, then there is no usable IDE device connected to the CPC. Single bits of this variable have the following function:

The bits 7, 6, 5 and 4 are reserved and set to zero.

Bit 3 = 0 --> CPC-IDE/SF2 device 0 missing (Master)
= 1 --> CPC-IDE/SF2 device 0 is ready to use

Bit 2 = 0 --> CPC-IDE/SF2 device 1 missing (Slave)
= 1 --> CPC-IDE/SF2 device 1 is ready to use

Bit 1 = 0 --> IDE8255 device 0 missing (Master)
= 1 --> IDE8255 device 0 is ready to use

Bit 0 = 0 --> IDE8255 device 1 missing (Slave)
= 1 --> IDE8255 device 1 is ready to use

IDE_RAM DS 1 ==> Contains the physical RAM select (I/O Port &7Fxx) of a 16 KB Exp.-RAM (&C4 to &FF). This RAM block will be used as hard disc cache. The usage of such a cache RAM speeds up hard disc access.

IDE_SECNT DS 1 ==> This variable defines the so called Sector Count
(but it is not used at the moment).

The next six variables address the actual 512 byte LBA sector of a hard disc (IDE) that is being used / shall be used now. When using LBA28 only four variables will be used: IDE_00_07 up to IDE_24_31, while the LBA48 mode uses all six variables IDE_00_07 to IDE_40_47. FutureOS can manage IDE hard discs up to 128 PB (Peta Byte) when using LBA48. 128 PB equals 134217728 GB.

```
IDE_00_07 DS 1      ;LBA28 uses IDE_00_07 to IDE_24_31 (4 bytes)
IDE_08_15 DS 1
IDE_16_23 DS 1
IDE_24_31 DS 1
IDE_32_39 DS 1      ;LBA48 uses IDE_00_07 to IDE_40_47 (6 bytes)
IDE_40_47 DS 1
```

Some of the above variables aren't used through the Desktop, but a program must deal with them. Only this way it's possible to create a cooperative OS.

Under FutureOS you can run some programs parallel, therefore they must look at the system-variables, especially the XRAM_?? variables!

Variables in the ROMs of the FutureOS

The ROM A of FutureOS contains at address &C114 the variable KOBYA.
Thereby KOBYA stands for 'KOntrollBYte A', that is ‘Control Byte A’
KOBYA serves the configuration of the FutureOS. Since KOBYA is located inside the ROM, you need to change ROM A to adapt KOBYA.

Which parameters are defined by OS variable KOBYA?

Bit 0 = 0 → Joystick 2 is used by the Desktop to move mouse arrow
= 1 * → No Joystick 2 usage in Desktop. Better usage of Hotkeys

Bit 1 = 0 → LambdaSpeak speaks as male, male voice
= 1 * → LambdaSpeak speaks as female, female voice

Bit 2 = 0 → SF2 RTC will be changed to BCD. OS faster!
= 1 * → SF2 RTC not changed! OS is compatible to other systems

Bit 3 = 0 → LambdaSpeak under FutureOS silent, no speech synthesis
= 1 * → LambdaSpeak speaks OS messages

Bit 4 = 0 → Normal / Compatible Mode, WiFi on, day saving time on
= 1 * → Health Mode, means WiFi off, no day saving time

The Bits 5 to 7 are currently not used.

The '*' marks preset options.
In FutureOS the ROM A variable KOBYA usually contains the value &1F = xxx1 1111

The variable KOBYB ('KOntrollBYte B', means ‘COntrol BYte B’) is located at address &C18B in ROM A. Its bits also define some configuration settings of the OS. Which parameters are now defined by KOBYB?

Bit 0 = 0 *=> MultiPlay Port 1 is connected to a Joystick
= 1 ==> MultiPlay Port 1 is connected to a Mouse

Bit 1 = 0 *=> MultiPlay Port 2 is connected to a Joystick
= 1 ==> MultiPlay Port 2 is connected to a Mouse

The Bits 2 to 7 are currently not used.

The '*' marks preset options. KOBYB contains the value &00 = xxxx xx00.

Selection of a particular FutureOS ROM A, B, C, D

The OS functions provided by FutureOS are split into four ROMs. They're called A, B, C and D.

If you want to use an OS routine, then you can use the API or you have to select the corresponding OS-ROM before calling the routine, if the correct ROM isn't selected. Intelligent programming can thereby save some microseconds.

Bank in / select one of the OS ROMs A, B, C or D using OS functions:

Select ROM A: `CALL OSRON_A` ;Bank in ROM A

Select ROM B: `CALL OSRON_B` ;Bank in ROM B

Select ROM C: `CALL OSRON_C` ;Bank in ROM C

Select ROM D: `CALL OSRON_D` ;Bank in ROM D

Select / Bank in an OS ROM using Z80 commands:

Select ROM A:
```
LD   BC,(&FF01) ;C = physical ROM number, B = &DF
OUT  (C),C      ;switch ROM A on
```

Select ROM B:
```
LD   BC,(&FF07) ;physical number of ROM B
OUT  (C),C      ;switch ROM B on
```

Select ROM C:
```
LD   BC,(&FF0D) ;physical number of ROM C
OUT  (C),C      ;switch ROM C on
```

Select ROM D:
```
LD   BC,(&FF13) ;physical number of ROM D
OUT  (C),C      ;switch ROM D on
```

The Application – Program – Interface (API)

The application programming interface (API) of most operating systems is quite slow. The fastest way to use an OS function is to directly call it. The API of FutureOS supports exactly this way of doing it. It's constructed in a simple and efficient way; deprived of all kinds of dead weight.

To call an OS routine in one of the four OS-ROMs you can go the direct way. Or you can use the API. The API provides a variety of entry points. You can call the following entry points, which select the correct OS-ROM for you: **ROM_A, ROM_B, ROM_C or ROM_D**.

They will switch on the new ROM, in which the OS function is located, then they call the target-routine (Z80 register HL contains the address of the OS function you want to call), finally they return. The new OS ROM stays selected.

The API entries ROM_S2T (S = source, T = target) select an OS-ROM for one target-routine-call, afterwards the source ROM is selected again. S and T can be one of the four ROMs A, B, C or D.

Example: The API entry ROM_A2C calls the target-routine in ROM C. ROM A is switched on after the end of the target-routine. The address of the target-routine must be provided in register IX.

Further information about the API is provided in file API-ENG.DOC. Please have a look there!

Aside the OS ROMs A, B, C and D FutureOS supports additional expansion ROMs (XROMs). An example is the IDE-ROM.

FutureOS eXpansion ROMs (XROMs)

Such ROMs are similar to expansion ROMs for the native OS. Between &FF00 and &FFFF they equal FutureOS ROM A. And between &FD80 and &FDE3 they contain functions to access OS functions of the FutureOS ROMs.
XROMs usually don't need an initialization routine, but can have one if they like. Refer to "API-X-EN.DOC" for further information.
FutureOS XROMs **always** (!) start with bytes &02, &09 at address &C000.
Please use only ROManager 2.16 (or later) to install XROMs, because they need their hard coded ROM numbers to be adjusted.

Program-Architecture

A BASIC/AMSDOS program has the maximum length of 42 KB. Under the 63K CP/M 2.2 or CP/M 3.1 (plus) a program may reach up to 62 KB. Bigger programs have to load separate overlays.

FutureOS programs can reach a length of up to 4 MB in one piece, its length is only limited by the connected expansion RAMs. But usually you should use a bit less, because FutureOS needs about 32 KB for buffering e.g. directories.

* There are two general kinds of FutureOS programs. First type is the main-memory-program. They're located in the first 64 KB (standard RAM, configuration &7FC0). A main memory program can reach a maximum length of up to 40 KB (&0000-&9FFF). If the program uses the file-tagging-bytes (FTBs) in addition it can be up to 44 KB. Never overwrite a byte above &B800 (System variables!). These main-memory-programs should end with the extension .64K. Start them with the RUN-icon.
Attention: The RAM-block from &0000 to &3FFF is used by many routines.
Furthermore there must be a charset between &3800 and &3FFF. Switch the lower ROM on to use the ROM charset.

* The second kind of a program is the expansion-memory-program. Its size only depends on the connected E-RAM. Therefore they can have a maximum length of 4 MB. Expansion-memory-programs should end with the extension .X16.
These programs have a special architecture. Normally there is a short loader-program, which examines if enough E-RAM is connected and where to load it. Then the main part is loaded into E-RAM. The corresponding RAM-variables XRAM_C4 ... _FF must be adapted; this is done by OS system calls (see later).
AKT_RAM must contain the right start E-RAM block, where the E-RAM program should be loaded. For example &7FC4 for the first E-RAM.

```
LD BC,&7FC4
LD (AKT_RAM),BC   ;if the program starts in block &7FC4 (1. E-RAM).
```

An expansion-memory-program is separated in 16 KB blocks; therefore bank-switching is required. The banking must be considered while coding.

At address &8000 you should copy a routine that manages the automatic switching to the next E-RAM. Example:

```
ORG  &8000              ;routine starts at &8000
PUSH AF                 ;save the registers
PUSH BC
LD   BC,(AKT_RAM)       ;get old E-RAM-status out of AKT_RAM
INC  C
SET  2,C                ;select the next 16 KB E-RAM block
OUT  (C),C              ;switch the new E-RAM on (between &4000 and &7FFF)
LD   (AKT_RAM),BC       ;write the new E-RAM configuration back in AKT_RAM
POP  BC
POP  AF                 ;get old register values
JP   &4000              ;jump to the start of the new E-RAM
```

The separation of the RAM brings some restrictions, like the length of a string; a string can't be located in two different E-RAM blocks. You can only access program subroutines of one 16 KB block at one time.
Theoretically you can switch complete 64 KB banks but then you have no access to the screen RAM!

Foreground- and Background-Programs

Independent of the fact if a program is a main memory program or a expansion memory program, all programs can be a foreground or a background program in addition.
What's a foreground-program? If a new program is loaded into memory it will become automatically the foreground program. The file-header of the program is copied to &BC00 by FutureOS. These 128 bytes from &BC00 to &BC7F contain the file header of the last loaded program or file (if a file header exists).
After the end of a foreground program it can be restarted using the RUN icon of the Desktop, without using the OK icon.
If you load/start a new foreground-program the header of the previous program gets overwritten with the new header. Therefore the old foreground program is lost.
To give a program the possibility to become resident in the RAM of the CPC, it can become a background-program.
Foreground- & background-programs are equal with one difference: The file header of an background-program is located between &BC80 and &BCFF. The program itself must copy its &80 byte header from &BC00 to &BC80 to become a background program.
But every program which want to install itself as new background-program has to check if another background-program is present (simply by checking the test-sum of the file-header). In this case the user has to remove the old background-program by ending/closing it.
Therefore every background-program must possess the feature of self-termination, than means to delete its file-header from &BC80 to &BCFF.

Shellstack: There is the alternative to move the header of the old background-program to a header stack. After the end of the new background program, the old header gets reinstalled. A shell stack can be created.

Interrupts

FutureOS works in Interrupt-Mode 2 (Vector-Interrupt-Mode). This interrupt mode is very powerful, because every external interrupt can branch to a distinct address in RAM. Therefore every user can easiely target his drivers for new developed hardware.

When starting FutureOS the interrupts are DIsabled, but can be Enabled when ever needed.

If a program wants to work with interrupt mode 1, then it can install its own interrupt routine at &0038. But it must switch the CPC to interrupt mode 1 (IM 1) before. The lower RAM must be selected, and a charset must be installed at &3800 in RAM. Using IM 1 it is absolutely necessary to switch the lower ROM off!

Before returning to the Desktop the Interrupt-Mode must be reset to 2 and the interrupts should be disabled (DI:IM 2). Furthermore the value of the I Register must be kept, it must be &BD (**`LD A,&BD:LD I,A`**). The I Register gives the highbyte of the interrupt-mode 2 jump-table.

The NMI (entry at address &0066) can be freely used by every program which wants to do it. Example: the planned 8-bit Inicron-network.

IV. The Desktop

After the start of FutureOS by typing !OS or !FDESK you get to the GUI (Turbo-Desk) of FutureOS. The screen in divided into three regions. In the upper part of the screen the icons are located; they are used to select storage media and functions. Below the file window is located, each page shows up to 64 file names. The keys Shift and Control are used to page up and down a page. The two lowest lines on screen are used as a status lines. The Desktop is handled hot keys or by a Joystick, cursor keys and copy, a mouse, lightpen, Grafpad II, the analogue joystick etc.

All menus entered by clicking an icon, can be left by pressing ESC.

Also entering numbers (in hexadecimal or decimal) can be stopped by ESC (or first DEL, then ESC).

Menu functions can be selected by pressing the corresponding number key or by moving the joystick or cursor keys up and down, confirm with fire button or copy key.

The storage device icons A...M:

Icons A...M shown in dark can be selected by a click. This activates the corresponding device. A second click on the now bright icon will deactivate it again. Hatched icons shouldn't be used, corresponding devices are not accessible. Press Space, then A-M to select a device.

The DIR icon (hot key D):

Clicking on this icon (hot key D) reads all DIRectories from all tagged storage devices. Subsequently the first page of the first DIRectory will be displayed on screen. The keys SHIFT and CONTROL scroll one page (of a DIRectroy) up and down.

To be able to perform file operations, the DIRs of the coresponding devices must be read first, since the OS buffers all DIRs in RAM.

If you try to read the directory of a disc that is not fitted into the drive, the OS will wait approximately five seconds, in this time you can insert the disc into the drive. Afterwards the drive will be deactivated and its icon will be shown hatched. In this case please insert the disc and click the DIR icon again.

If the automatic-DIR function is activated then the DIRectories will be read automatically as soon as you click at one of the device icons.

You can activate Auto-DIR in the OS configuration using 'Config OS'.

The selection of files:

If a page of filenames of a directory is currently shown on screen, then the filenames can be tagged by using the mouse pointer. Tagged files are shown underlined. If you click at a already tagged filename, then the filename will become untagged again. Using FutureOS all file-operations deal with tagged files only.

The TYPE icon (hot key V):

To display files on the screen, first you have to tag the filenames.

Then you must click on the TYPE icon or use hot key V. Press the "1" key for TYPing files on the screen or to show a picture. First the file-header is shown (if exists). Press a key. Then the screen will be shown or a second menu appears. Here you can select the screen format (columns and lines) in which the first text-file should be displayed. The "1" selects the usual 80 chars per line and 25 lines per screen. The OS then loads the file and shows its first page on the screen. You can scroll a page up and down by using the cursor keys or a joystick (up and down). To exit TYPE simply press COPY or ESC. Or scroll down to the files lower end. If a picture is shown use the arrow keys to select the screen's MODE and format. If you have tagged more files the same procedure will be repeated for every file, beginning with displaying the header.

Menu point 2 is used to show the 128 bytes file-headers of a tagged file. After showing the first file-header, type SPACE to show the file-header of the next tagged file (if there is one). If you press the ESCape key the function is stopped and the last file remains tagged. Therefore it can be started directly with icon RUN.

The LOAD icon (hot key L):

Click the LOAD icon to load the first tagged file. The displayed menu decides in which way you want to load the file.

Loading type 0: Loads a file according to its file-header. The target address can be located anywhere in (expansion) RAM.

Loading type 1: Loads a file in the standard 64 KB RAM at address &0000. Such a file often has the extension "64K".

Loading type 2: Loads a file in the expansion RAM. The start address is &4000 in the first 16K expansion RAM block (&7FC4). The file can contain up to 4 MB data depending on the E-RAM. Such files often end with extension "X16".

Loading types "3" and "4" load files at any address in the standard RAM (type 3) or in the up to 4 MB expansion RAM (type 4).

- All values must be entered in hexadecimal numbers.
- The expansion RAM select must be provided as a physical value.

The SAVE icon (hot key S):

It allows you to save previously load files or parts of the memory as files. You can only save to an active drive, whose directory has been read before. Clicking the SAVE icon provides four ways to save a file:

1 Save Foreground program: Saves a previously loaded file or started program. The target device, user number, filename and the file extension of the program/file can be edited.
The target device is symbolized through the character A-M.

2 Save Background program: This function is equal to point 1 but saves the actual Background-program.
You can save a Background-program only if you've started one before.

3 Save main memory: Saves a file from a defined 64 KB block of RAM. You will be asked for drive, user, filename and extension by editing the string "A00:ProgNameExt". The start address has to be entered in hexadecimal numbers. If you want to save from the first 64 KB of the main memory then provide the value &C0, when being asked for the E-RAM configuration "Source Block". Finally you have to provide the length of the file in hexadecimal numbers and full KB. The maximum file length should be not more than 64 KB.

4 Save expansion-RAM: This function can be used to save a file from the expansion RAM. You can choose every kind of valid physical RAM configuration when being asked for it.
The file length depends only on the connected expansion RAM.

- All values must be entered in hexadecimal numbers.
- The expansion RAM select must be provided as a physical value.
- The length of a file must be provides in KB. &0010 means 16 KB e.g.

The OK icon (hot key O):

It provides the possibility to jump back to a previously started application, if this feature is supported by the application.
An application can jump to the Desktop, there files can be tagged e.g., subsequently the click on OK jumps back to the application.
If you click the OK icon while no application uses it, an error message will be displayed.

The <I> icon (hot key I):

If a information system has been started previously, the I icon will call it. The I icon works similar to the OK icon.
If not information system has been initialized the click of the I icon will result in an error-message.

The ERAse icon (hot key E):

It has different functions, which are all connected with erasing data. After the activation of the ERA icon you can choose a function in the appearing menu:

1. Erase file(s): By pressing "1" you'll enter the file eraser. It allows you to erase all tagged files. Different drives can be used.
You can then press "4" or "5" to choose the erase mode.
With 4) All & now you erase all tagged files immediately. The erased files are then gone forever. If you aren't sure use 5) instead of 4)!
With 5) With security-query you can delete files selective. You will be queried for every file if it should be erased or not.

2) Format disc: allows you to format discs. You can format Data-, System-, IBM- or Vortex. After choosing the format you will be asked for the drive in which the disc is located. Please put first the disc in the correct drive, then press the drive-letter "A".."H" and RETURN.
After that you will be asked for "Double Steps" and if you want to choose the tracks to be format. Just press RETURN for standard parameters.

The REName icon (hot key N):

After clicking REN press "1" to REName files, only tagged files are used. After pressing 1 the first tagged filename is displayed twice.
The upper one remembers you how it looks like before renaming. The lower filename is there to be renamed. You can change the user-number, the file-name and the file-extension. But you can't change the drive.
You can only use chars between 32 and 127 inclusive for the new name.

The COPY icon (hot keys C or F):

It allows you to make COPY of discs or files. The menu has two points:

Point 1 gives the possibility to copy files multidirectional! That means that there can exist multiple sources and multiple targets of files. In the second menu you can decide if you want to copy all the tagged files to one target (point 3) of if you want to copy different files to different targets (point 4). If you copy files with point 4 you can not only choose the destination. You can edit the user number, file-name and extension, too.

Point 2 of the COPY icon main-menu allows you to copy a disc. The source disc can have Data-, System, IBM- or Vortex format. You can use both sides of the disc and/or discs with double-step.
If the source and the target drives are identical you have to swap the discs. At the bottom of the screen will be messages. The disc-format is emerged automatically. Depending on the format the target-disc may be formatted.

The PRINT icon (hot key P):

It has two features. The printing of DIRectories or files on the connected Printer. Hence the icon-menu has two points:
1. Print file(s): allows you the printing of all tagged files one after one. The ESC key stops it.
2. Print DIR: prints out the DIRectory of a drive. Just type in the drive letter and press RETURN.

The RETAG icon (hot key A):

Its function is the retagging of all used (previously tagged) files shown with a line through the filename. These files will become tagged again (shown underlined).
Through the RETAG icon you can use a subset of files for different operations. For example you can first type them and then rename their filename or whatever you want.

The UNTAG icon (hot key U):

Its function is to UNTAG all tagged files. After a click on this icon all previously tagged (and old-tagged) files will be untagged. All files will be untagged.
You can use the DIR icon to get the same effect (if you haven't changed a disc). The DIR icon is slower than the UNTAG icon, because of the need to read all DIRs again. Please use the UNTAG icon instead of the DIR icon when dealing with empty DIRs!

The MONitor icon (hot key M):

It's used to jump in the little machine monitor of the FutureOS. There you can examine the RAM, test Routines and so on. More information is given in chapter V.
Please use the monitor only if you are very familiar with your CPC.
One wrong value can crash the system.

The ALARM / WECKER icon (hot key W):

Use the Cursor-keys and COPY (Joystick and Fire 1) or ESC to set the Alert time. Ignore this icon if you have no smart or Real Time watch.

The END icon (hot key Q):

1. Start FutureOS new starts the Future Operating System again. This is nearly the same if you had switched on your CPC and type |OS.
2. Initialize AMS-OS ends the current FutureOS session and initializes Amstrads OS. This function is equal to a reset of the CPC.

The RUN icon (hot key X):

Its action depends on the existence of a tagged file. If there are tagged files the first tagged file will be load into memory and it will be executed there. But only if this file has a file-header, else an error message appears. The first tagged file will be used. If there is no tagged file present the following menu is displayed:

1. Start actual Foreground-program. The last loaded program will be started again, because it's still in the RAM and the OS has memorized its location through its file-header.

2. Start actual Background-program. A Background program is a special program which is nested in the expansion memory.

3. Jump into RAM. This point allows you to jump at any address in any connected (expansion-)RAM. When you want to jump in the standard memory you should give &C0 as RAM block when you're asked.

4. Execute ROM application. First you select an FutureOS expansion ROM and then you select an application of that XROM, it will be executed.
All the XROMs shall have consecutive ROM numbers, to be able to use "Up" and "Down" to select the desired XROM.

The TIME and DATE icons (hot keys Y and Z):

The TIME icons are at the left side and the DATE icons at the right side. You can use them to read of set time and date. But please only use them when a smart watch is connected. For changing the values just use the Cursor-keys and Copy (or a Joystick and Fire 1). Press the ESCape key to leave.

The IDE icon (hot key H):

It allows to access IDE devices connected through the X-MASS, IDE8255, CPC-IDE, Albireo or the SYMBiFACE II. To use IDE devices a fifth FutureOS ROM (IDE ROM) is needed, which contains IDE/FAT management software.

Additional hot keys

- With "j" (jump) the mouse pointer will be set to the first file name in the file window.
- With "b" (bypass) the mouse pointer will be moved to the next file name, but without tagging / untagging the file underneath. Using hot keys G and B allow to tag / untag all files and then launch an OS function.

V. The Machine-Monitor

The machine-monitor provides a very powerful tool. You can access every bit in RAM, read/write 16 bit I/O addresses, program the hardware due to this and call OS routines. If you want to start/test an machine routine you can use the machine-monitor or the RUN icon.

Using the monitor you can set ALL Z80 registers before calling a routine. But please use the monitor only if you know your CPC and its features very well. At all important stages of a function you can stop with the ESC key. Please enter values in the hexadecimal system.

d = FutureOS - <d>ump/edit system

This function displays a part of the memory in hexadecimal and ASCII. First you're asked for a 16 bit start-address. After entering this address, &01E0 bytes will be displayed. 16 bytes are shown in each line. Hexadecimal values are shown left and the ASCII characters at the right side. Use the Cursor UP / DOWN keys to show the last or next page.

If you press the key "e" then you enter the Edit-mode. Every other key brings you back to the menu of the monitor. The cursor is shown at the upper left position in the editor mode. You can move the cursor with all four cursor-keys. If you want to change a byte, just press the "COPY" key, then you can enter the new hexadecimal value. Finish with Return or quit with ESC. To leave the Editor-mode you have to press the little Enter-key.

The parameters selected with function "c = RAM/ROM <c>onfiguration" are considered by the dump / edit functions.

p = FutureOS - <p>orting system

This feature allows you to send bytes to I/O addresses or to get bytes from an I/O address. The cursor can be moved with the cursor-keys. ESC brings you back to the menu of the monitor.

If you want to read data from an I/O address, just move the cursor to the corresponding address ("Inp." column) and press COPY to read a byte from this I/O address.

To send a byte to an I/O address you have to move the cursor to the "Outp" column and press COPY. Now you have to type in the hexadecimal value you want to output. Finish it with RETURN and the value will be sent immediately. ESC interrupts.

e = <e>dit all Z80 registers

This point is used to change the values of the Z80 registers. If you call an routine with the machine monitor then the registers of the Z80 are preloaded with the values entered here.

All Z80 registers are shown with their values. Use the cursor-keys to go to the register you want to change, then press COPY and enter the new hexadecimal number. Press RETURN to end or ESC to abort. To leave just press the little Enter key. The values edited here are stored in the 16 bit RAM-variables REG_AF to REG_PC. Look before.

If a routine returns to the machine monitor these editable RAM-registers contain the values of all Z80 registers.

If a routine jumps to the OS function CARET, the Z80 registers will be saved in this RAM registers and you enter the menu of the machine monitor. This way you can analyze the return conditions of any machine language routine.

c = RAM/ROM <c>onfiguration

Here you define if you want to work with RAM or ROM and which RAM configuration should be used. All other functions of the monitor take care of these settings.

First you're asked if the lower ROM (it contains the charset!) should be selected. Type "y"es or "n"o. If you select NO then the lower RAM is active.

Afterwards you're asked for the physical RAM-configuration. But use only the following values: &C0 (standard 64 KB) or &C4-&FF (one out of 32 E-RAMs). The selected RAM is switched in between &4000 and &7FFF.

If you own a CPCplus and the Plus version of FutureOS then you can switch the Memory-Mapped-I/O on or off (y or n).

The MM-I/O is switched on between &4000 and &7FFF instead of RAM. The MM groups have highest priority.

m = <m>ove memroy block

With this feature you can copy a memory block of any length to any address in RAM.
First you're asked for the start-address, it's the first byte of the block you want to move.
Then you're asked for the end-address of the block, that's the last byte of the block (included).
Third you have to enter the target-address of the block, which is where the block should be copied to.
Attention! It's NOT allowed to overwrite the RAM between &B800 and &BFFF. It contains the OS system RAM and OS variables!

i = <i>nitialize memory block

This function allows you to initialize a RAM-block. First you enter the start (first byte of the block) and end-address (last byte of this block, inclusive). Then you're asked for a 16 bit value. This value is used to fill the defined block.

r = call an mc <r>outine

With "r" you can call machine code "r"outines. The Z80 registers are loaded from the variables REG_AF - REG_PC previously. See before!

You will be asked for the start-address of the routine, this address is the entry point of the routine. After entering the start address press RETURN to call the routine or press (DEL and) ESC to stop this function.

The RAM and the RAM will be selected like defined before, by using function "c"onfiguration.

x = FutureOS - e<x>it the MONITOR

This function is used to quit the machine monitor and to return to the Desktop. You can't leave the monitor through (unwanted) pressing of ESCape.

VI. Accurate explanations about specific FutureOS features

Managing the keyboard

The keyboard manager of FutureOS is organized in four levels. The keys of the keyboard are only scanned if needed. The two digital Joysticks are treated as a part of the keyboard.

While BASIC scans the keyboard 50 times in a second, FutureOS will scan the keyboard only on demand. This method saves some CPU time. If the state of some or all keys shall be investigated you can call the corresponding OS function, which delivers the state of the keyboard. Later it will be shown that there is a broad variety in key scanning.

Under BASIC you work with three keyboard levels (normal, shift or control). FutureOS has four levels, in the fourth level shift and control are pressed both in parallel.
There is a restriction in the combined SHIFT+CONTROL level. Some keys can't be used in this level. The construction of the keyboard hardware itself is responsible for this. The following nine keys can't be used:

"SPACE" ; "DEL" ; "f0" ; "f." ; "," ; "." ; "V" ; "X" and "Z"

If you connect an external keyboard to your CPC you can usually use all keys in the Control+Shift level.

What is the advantage of using a fourth keyboard level? Well, the four keyboard layers allow dispersing all possible 256 characters to the 80 keys of the keyboard. If there would be only three keyboard levels, it would be possible to display only 3 * 80 = 240 characters. But not all 256 characters of a 8 bit charset.

The key occupancy

A table of 80 bytes of the OS system RAM is reserved for each of the four keyboard levels. These areas equal the keyboard matrix in each case.

By changing a byte of this table it is possible to assign a defined character to a defined key.

The table of the NORMAL --------- level starts in RAM at	TAST_N	=	&B980
The table of the SHIFT ---------- level starts in RAM at	TAST_S	=	&BA00
The table of the CONTROL -------- level starts in RAM at	TAST_C	=	&BA80
The table of the SHIFT+CONTROL level starts in RAM at	TAST_SC	=	&BB00

A ROM copy of the four keyboard tables is located in the FutureOS ROM C at address XAST_N = &FDAA. If you change a byte there, then this exchange will be permanent in all FutureOS sessions. But be careful, don't lose important keys.

OS functions of the keyboard manager

Under FutureOS a variety of OS functions can be used to manage the keyboard. You should refer to the file '**API-A-EN.DOC**' . The following OS functions are documented in detail in this file. One useful OS function is **H_XALLET**. It provides the ASCII character that corresponds to the key pressed at this moment. The SHIFT, CONTROL and CAPS keys are considered. More OS functions are provided:

Important OS functions for the keyboard management

H_XALLET	scans keyboard, CAPS-LOCK, Shift, Control states are regarded
HOLE1TS	scans if a particular key is pressed or not
H_JC	scans both joysticks, cursor keys (+Copy, little Enter) and the MultiPlay H_JC is bit compatible to OS functions H_JOY and H_CCE
M_CON	scans proportional mice (SYMBiFACE 2, SF3, and MultiPlay) returned result (in A) is bit-compatible to H_JC, H_JOY, H_CCE
H_JCM	equals a combination of H_JC and M_CON. Scans all HID devices
XWART	like WART_TS, but has defined first and second waiting times
WATA	waits until a key is pressed
RB_8	8 bit input, depending on numbering (hexadecimal or decimal) this OS function either jumps to BIT8_IN or B8DIN
RB_16	16 bit input, depending on numbering (hexadecimal or decimal) this OS function either jumps to B16IN or B16DIN
STED	shows and edits a string. This string has a 16 bit length

Further OS functions for the keyboard management

H_ALLET	scans keyboard, but the Caps-Lock state is NOT regarded
TST_TS	scans if any key is pressed in common (Yes/No)
WART_TS	waits until no key (or joystick) is pressed any longer
HOLETST	reads the state of all 80 keys into RAM
A_F_0_9	scans only the keys 0-9 and a-f from the keyboard, pressed keys are displayed in mode 2 on the screen (at C_POS)
H_CURA	scans only the four cursor keys and the copy key
H_CCE	scans cursor keys, copy & little enter. Joystick compatible
CUR_CPY	scans cursor keys, copy and the ESC key
H_JOY	scans both digital joysticks. Bit compatible to H_CCE
H_CS	scans the SHIFT and CONTROL keys
BIT8_IN	is an input routine for 8-bit values, hexadecimal
B8DIN	is an input routine for 8-bit values, decimal
B16IN	is an input routine for 16-bit values, hexadecimal
B16DIN	is an input routine for 16-bit values, decimal

For a regular application H_JCM or H_JC, H_XALLET, XWART, WATA and STED are sufficient.

The keyboard matrix of the CPCs

////	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
l0 :	f.	ENTER	f3	f6	f9	Cur.D	Cur.R	Cur.U
l1 :	f0	f2	f1	f5	f8	f7	Copy	Cur.L
l2 :	CTRL	\	SHIFT	f4	]	RET.	[	CLR
l3 :	.	/	:	;	P	@	-	^
l4 :	,	M	K	L	I	O	9	0
l5 :	SPACE	N	J	H	Y	U	7	8
l6 :	V	B	F	G	T	R	5	6
l7 :	X	C	D	S	W	E	3	4
l8 :	Z	Caps	A	TAB	Q	ESC	2	1
l9 :	DEL	fire3	fire2	fire1	Joy.R	Joy.L	Joy.D	Joy.U

Explanations: Joy. R = right, L = left, D = down and U = up.
The "f." is the point in the block of function keys.
l0 - l9: line of key matrix.

Chars and Strings

BASIC provides one central OS call to display characters. This routine is very mighty, it works in all screen modes, allows different print modes and more. The price you pay is that BASIC's PRINT routine is relatively slow.

FutureOS provides different ways to display chars, strings or terms. Terms are strings that contain control codes. Characters can be displayed in Mode 1 and 2. This specificity makes FutureOS up to 80 times faster than BASIC when displaying a character or string.

Attention: Every OS function used to display chars or strings needs a charset between &3800 and &3FFF. You can switch the lower ROM on or you can load a software character set at address &3800 in RAM. Look at the RAM-variable: RAMCHAR (&B847).

The control codes of the Future Operating System

You can define all control codes of the FutureOS very freely. But only the first 32 chars are control codes (0-31 / &00-&1F). These 32 chars are only used as control codes if a terminal routine is used to display a chain of characters, like TERM_2, TER_BB ...

Which control codes FutureOS uses in which way and for which function will be discussed in detail on the following pages.

If you want to emulate an alien terminal (eg. ANSI) or if you want to implement your own routine you can easily change control codes.

For screen Mode 1 and 2 a own control code table exists for all 32 control codes. Therefore the same control code can have different functions depending on the screen mode 1 or 2. Each table contain 32 variables (16 bit). Every variable contain the address of a routine called by the corresponding control code. Both tables are 32 * 2 byte = 64 bytes in length.

Control code table for Mode 1 starts at: TAS_S1 = &B800 to &B83F
Control code table for Mode 2 starts at: TAS_S2 = &B900 to &B93F

These tables start with control code &00 (00) and finish with code &1F (31). Every 16 bit variable consists of the low-byte first, followed by the high-byte of the address of the control code routine.

Features of the Mode 1 control codes

The following information is provided for Mode 1 control codes, some Mode 2 control codes are used differently. Control codes which provide equal features in both modes are described in detail only in "Features of the Mode 2 control codes" (see later).

Code &00 = 00 : End of a string, look Mode 2.

Code &01 = 01 : Select character set (&3800-&3FFF) of RAM (lower-RAM).

Code &02 = 02 : Select character set (&3800-&3FFF) of ROM (lower-ROM).

Code &03 = 03 : Select E-RAM between &4000 and &7FFF, look Mode 2.

Code &06 = 06 : Half a space, the cursor position is moved right for half space. This allows including half spaces between Mode 1 chars.

Code &08 = 08 : display x spaces. After code &08 the number of spaces (x) follows. x can have a value between 1 and 127.

Code &09 = 09 : TAB (1-8 chars), look Mode 2.

Code &0A = 10 : LINEFEED without Return, look Mode 2.

Code &0B = 11 : Clear screen, like the BASIC command CLS.

Code &0C = 12 : Cursor home, most up left position, look Mode 2.

Code &0D = 13 : RETURN without Linefeed, look Mode 2.

Code &0E = 14 : real RETURN, equals the Codes &0A+&0D, look Mode 2.

Code &0F = 15 : set new text-address, where string continues, look M2.

Code &10 = 16 : Switch to Pen 1. All characters following after this control code will be printed in Pen 1.

Code &11 = 17 : Pen 2 is now standard.

Code &12 = 18 : Pen 3 is now standard.

Code &13 = 19 : Mixed Pen 1 & 2 color is now standard. All chars have two colors. This is the fastest way to print a Mode 1 char on screen.

Code &1A = 26 : Equals Code 0. End of a string, look Mode 2.

Code &1D = 29 : Zoom some chars * 8, look Mode 2.

Code &1E = 30 : LOCATE 32 * 32, moves cursor at any new position on the screen. Two bytes must follow after the Code &1E. The first byte defines the new Y-position (0..31) of the cursor, the second contains the new X-position (0..31).
This control code is used only in 32x32 mode, when the screen is set to 32 chars (Mode 1) per line and 32 lines per screen. You can set the screen to 32x32 mode with the OS function S64X32.

Code &1F = 31 : LOCATE 40 * 25, the cursor can be moved to any new position on the screen. Control code &1F is followed by two bytes, first the Y-position (0..24), then the X-position (0..39).
This control code is used only if the screen has 25 lines and 40 characters per line. You can switch the screen to 40 * 25 f.e. with the OS function S80X25.

Features of the Mode 2 control codes

Code &00 = 00 : End of a String. Every string must have a end, the end of a string is marked with a byte &00. (code &1A has the same sense)

Code &01 = 01 : switch lower RAM on, a charset must start at &3800.

Code &02 = 02 : select lower ROM, the ROM charset is used now.

Code &03 = 03 : Selects an 16 KB expansion-RAM between &4000 and &7FFF. After code &03 one byte follows, that selects a RAM block of the first 512 KB E-RAM. This byte contains the physical RAM select, which can be &C4, &C5, &C6, &C7, &CC, &D4 ... &FF or &C0 (switch first 64 KB on). This control code allows to show a string anywhere in the E-RAM.

Code &04 = 04 : show vertical x times the char y in 80 * 25 mode. Code &04 displays a char (y) x times one beneath the other. Two bytes follow after code &04. The first says how often the second byte should be displayed. The cursor position is moved only one position to the right. This code is used if the screen is set to 80 characters per line and 25 lines (see OS function S80X25).

Code &05 = 05 : shows a char (y) for x times vertical in 64 * 32 mode. Two bytes follow after code &05. First the number of characters that should be displayed. Second the char itself. The chars are printed one beneath the other. The cursor is moved one position right. Use code &05 with 64 chars/line and 32 lines/screen (see OS function S64X32).

Code &06 = 06 : moves cursor one position to the right (SPACE).

Code &07 = 07 : display char y for x times horizontal. Code &07 prints a char for x times, after code &07, three bytes follows. First the low and second the high byte of the number of bytes that should be shown. Third the character which should be displayed.

Code &08 = 08 : print x spaces on screen. Code &08 is followed by one byte which contains the number of spaces (1..255) which should be added to the actual cursor position.

Code &09 = 09 : TAB, the cursor is set right to the next TAB position. A TAB position exists every eight character positions.

Code &0A = 10 : LINEFEED, the curser is moved one line down, but he is NOT moved to the left end of the line, NO RETURN is made.

Code &0B = 11 : the screen is cleared, equals BASIC command CLS.

Code &0C = 12 : Cursor home, the cursor is set to the left upper position of the screen. The following character will be shown there.

Code &0D = 13 : RETURN, the cursor is set to the beginning (left) of the actual line. But NO Linefeed is made!

Code &0E = 14 : real RETURN, the cursor is set to the left border of the next line below. Code &0E equals Code &0A and &0D.

Code &0F = 15 : set new text-address. Code &0F is followed by two bytes, which contain the new address where the string is continued (first the low byte, then the high byte). Using code &0F you can jump from one string into another.

Code &10 = 16 : all following characters will be displayed normal now, without any attributes.

Code &11 = 17 : the chars will now be shown inverted.

Code &12 = 18 : following chars are shown in italics.

Code &13 = 19 : all following chars are now underlined.

Code &14 = 20 : now all chars are printed crossed out.

Code &1A = 26 : End of the string. This is the same as code &00, it is kept for compatibility with the ASCII code.

Code &1D = 29 : Zoom some chars (* 8). Code &1D is followed by one byte which contains the number of chars to zoom eightfold. The number of zoomed chars should lie between 1 and 8, depending on the cursor position.

Code &1E = 30 : LOCATE 64 * 32, the cursor is set at a new position on the screen. Code &1E is followed by two bytes, the first contains the Y-position (0..31), the second contains the X-position (0..63).
This code needs the screen to be set to 64 columns per line and 32 lines per page. Use OS function S64X32 to set 64 * 32 mode.

Code &1F = 31 : LOCATE 80 * 25, the cursor is set new. Code &1F is followed by two bytes. The first one gives the Y-position (0..24), the second one contains the X-position (0..79). This code need the screen mode set to 80 columns per line and 25 lines per page; use OS function S80X25 to set 80 * 25 mode.

New definition and redefinition of control codes:

The FutureOS allows redefining the function of every control code. If you change a control code, please take care of the conventions. They must be uphold faithfully.

But how to give a control code a new function? This is shown in the following:

- Every control code has two bytes in RAM twice. Two bytes for Mode 1 and two bytes for Mode 2. These two bytes contain an address (first the low byte, second the high byte). This address points to the routine which is called through the control code.
- If a control code is processed, its corresponding address is read and the Z80 jumps to this subroutine (mode dependent).
- To (re)define a control code you have to change the address in the control code table. The tables are located in the system RAM at address TAS_S1 (Mode 1) and TAS_S2 (Mode 2). The tables start with the address of the routine processing control code &00.
- Now the control code is new defined.

Attention: The routine for a control code is NOT allowed to change the ROM state! ROM A must remain active! If a control code would change the ROM configuration, the system can crash.
Furthermore the initial content of register DE must be copied to HL after the end of the control code routine. If DE was not used, you can simply use to code EX DE,HL at the end. If you want to use DE too, look at the listing below...
If the control code is followed by some parameter bytes you can read them beginning at the address in register DE. Else DE contains the address of the next byte of text. Look listing below...

Example:
To give control code &00 under Mode 2 a new function, you only have to put a new target address at &B900 (first lowbyte, then highbyte).

The control code table for Mode 2 starts at &B900.

If you want to redefine code &00 for Mode 1, you have to put the target address at &B800 (not &B900).

Formula: control code * 2 + &B800 or &B900

Example for a new control code general:

```
ConCode PUSH DE   ; save DE on the stack

 ???????????      ; now the control code subroutine follows

 POP HL           ; load old value (from DE) into HL!
 RET              ; and back.
```

Example: Control Code subroutine to set the border colour:

```
SET_BOR LD BC,&7F00 : OUT (C),C     ; select Border color register.

 LD  A,(DE) : INC DE                ; get new color, it follows cont.code
 OUT (C),A                          ; set the new Border color active.
 EX  DE,HL : RET                    ; transfer DE to HL and back.
```

If you want to reset both the control-code-tables (like they have been after the start of the OS) just call OS function CSTI in ROM A.

RAM variables connected with chars and strings

* First you should know **C_POS**. C_POS contains the actual cursor position. Normally it contains a value between &BFFF and &FFFF. C_POS is a direct pointer into the screen- / video RAM.
The pointer in C_POS has a screen mode dependent displacement. In MODE 2 C_POS contains a value one smaller than the real cursor position. In screen mode 2 you have to add two bytes to C_POS for the real position of the cursor. The reason is that the char printing routines first increase C_POS and then print the next char. In Mode 1 double width chars are used, therefore you have to add two to C_POS.

* The variables **MAX_CRX** and **MAX_CRY** contain the number of columns per line (MAX_CRX) and the lines per screen page (MAX_CRY). Both variables contain Mode 2 compatible values. The real screen mode doesn't matter.
Example: Screen Mode 1 is active, the screen has 25 lines, and every line has 40 chars. Then MAX_CRX contains the value 80, because of its Mode 2 compatibility. In detail MAX_CRX contains the screen bytes per line.
Both variables are used by some control codes. Therefore they must keep correct values. The OS functions S80X25, S68X30 and S64X32 set MAX_CRX and MAX_CRY correctly.

* RAMCHAR decides if a ROM or a RAM charset is used, that means if the lower ROM is switched on or off.
If Bit 2 of RAMCHAR contains the value 0, then the lower ROM is switched on, the ROM charset will be used.
But if Bit 2 of RAMCHAR = 1 then the RAM charset is used. Then there must be a RAM charset between &3800 and &3FFF!
The bits 0 and 1 or RAMCHAR contain the screen mode 0, 1, 2 (or 3).

Example code: (these lines should stay at every program start)

```
LD  BC,&7F81      ;switch MODE 1 on // (or use LD BC,&7F82 for Mode 2)
LD  A,(RAMCHAR)   ;RAM or ROM charset ?
AND A,&04         ;Bit 2 isolated! If set => then switch lower RAM on
OR  A,C           ;Bit 2 and &81 (or &82 for Mode 2) are connected
OUT (C),A         ;switch lower RAM or ROM on, they contain the charset
```

Printing single chars on the screen:

* Mode 1: Under screen mode 1 you can print chars in every one of the three pens. Furthermore you can display chars in a color mix of the pens 1 and 2, this is the fastest mode 1 display routine.

PRI0GG ==> display a char with pen 1 (normally yellow).
PRI0BB ==> display a char with pen 2 (normally blue).
PRI0RR ==> display a char with pen 3 (normally red).
PRI0GB ==> display a char with pen 1 and 2 (yellow, blue).

* Mode 2: Under screen mode 2 you can print chars with attributes:

PR_2 ==> displays a char without any attribute.
PR_2I ==> displays a inverted char.
PR_2K ==> displays a char italic.
PR_2U ==> displays a char underlined.
PR_2D ==> displays a char streaked out.

All these OS functions (located in ROM A) display a control code as a regular character on the screen. Neither in Mode 1 nor in Mode 2.

Display a string with defined length on the screen

A string is a chain of chars with defined length. Control codes are displayed as characters. The corresponding OS functions are located in the ROM A of FutureOS.

* Mode 1: Strings can be displayed using all three pens and using the mixed pen 1+2 color:

STR_GG ==> Pen 1 (normally yellow).
STR_BB ==> Pen 2 (normally blue).
STR_RR ==> Pen 3 (normally red).
STR_GB ==> Pen 1 + 2 (normally yellow, blue).

* Mode 2: Strings can be displayed with all five attribtues:

STR_2 ==> normal
STR_2I ==> inverted
STR_2K ==> italic
STR_2U ==> underlined
STR_2D ==> streaked out

Display Terms with variable length, and control chars

A Term is a string of variable length, a chain of characters which is terminated using the byte &00 or &1A. All 32 control codes are treated as control codes.

Like usually you have to take care that there is a charset located at address &3800. Or switch the lower ROM on and use the ROM charset.

* Mode 1: Terms can contain chars using all pens. The mixed-pen characters can be used too. Control codes can switch the pen.

TER_GG ==> Pen 1
TER_BB ==> Pen 2
TER_RR ==> Pen 3
TER_GB ==> Pen 1 + 2

Mode 2: You can use all five attributes and switch between them by using control codes.

TERM_2 ==> normal
TERM_2I ==> inverted
TERM_2K ==> italic
TERM_2U ==> underlined
TERM_2D ==> streaked out

With PR2GR you can display 8-fold zoomed giant characters.

Floppy-disc management

The mass storage manager of FutureOS was programmed with high efforts. The FDC- and HDC- management of FutureOS is compatible to AMSDOS and to the usual floppy disc formats like Data, System, IBM and Vortex. Despite the complete compatibility the floppy disc management is organized in a different way, compared to AMSDOS or CP/M. Nearly all OS functions for FDC management are located in ROM B and in some parts of ROM C. Compared with AmsDOS the FutureOS disc-manager is expanded. Examples are the file-headers of non-ASCII files and the management of up to eight floppy disc drives at the same time. Four drives can be connected to the internal FDC. Under FutureOS this four drives are named A, B, C and D. To be able to use C and D a little hardware patch is needed. The internal FDC is the FDC located inside the 6128. It's base-address (main-status-register) is &FB7E.
Four floppy disc drives can also be connected to the external FDC. They are termed E, F, G and H. The external FDC is the FDC from the F1-D and F1-S floppy drives produced by the company Vortex, Its base address is &FBF6.

The architecture of the disc-manager

The FutureOS disc-manager is completely different organized than the one from AmsDOS or CP/M.
Under FutureOS all the read DIR(ectories) of the drives are buffered in RAM. You have to read the corresponding DIRs before starting file-operations. The RAM buffering of DIRs speeds up all floppy disc operations. If you write to disc, or read from disc the read/write-head has only to step to the data-tracks of a file. It has NOT always to step back to the DIR. Combined with the fast track-load (or track-write) routines the disc-manager is several times faster than AmsDOS. Furthermore the DIR is only written once if you use a mass-operation like ERAse, RENname etc.
If AMSDOS loads a file, it loads one block after another. FutureOS first creates a Track/Sector table of all blocks of a file. Then the file will be loaded at once. Writing a file is done the same way. In addition FutureOS uses no head-load-time. After starting the drive-motor, the drive is used after it reports to be ready. While the OS waits for the drive to be ready, the Track/Sector table is created, no time is lost.
Every drive has its OWN Step-rate-time. If you connect a faster drive, you can adapt its step rate time in RAM or ROM. The step rate time of all eight drives (A..H) is located at DSWZ = &BA50 in RAM and at RSWZ = &C017 in ROM B of FutureOS. These bytes are organized like shown:

&00 ==> 32 ms step rate time	//	&10 ==> 30 ms step rate time
&20 ==> 28 ms step rate time	//	&30 ==> 26 ms step rate time
&40 ==> 24 ms step rate time	//	&50 ==> 22 ms step rate time
&60 ==> 20 ms step rate time	//	&70 ==> 18 ms step rate time
&80 ==> 16 ms step rate time	//	&90 ==> 14 ms step rate time
&A0 ==> 12 ms step rate time	//	&B0 ==> 10 ms step rate time
&C0 ==> 8 ms step rate time	//	&D0 ==> 6 ms step rate time
&E0 ==> 4 ms step rate time	//	&F0 ==> 2 ms step rate time

Standard 3 inch drives can run with 12 ms (few 10 ms). Newer 3.5 or 5.25 inch drives (80 tracks) run with 4 ms. The lower the step rate time is, the faster the disc access.

OS functions of the FDC management:

FutureOS provides a variety of OS functions for disc access. The low-level functions can be used to gain direct access over the (two) FDCs.

There are functions to read (or write) tracks and DIRs to (from) disc or hard-disc. High-level functions allow you to read/write a file, to format discs, read file-headers etc.

The FDC- and mass storage management of FutureOS are discussed in detail in the files API-B-EN.DOC and API-C-DE.DOC. For the standard usage take a look at OS functions LADEN, LADE_N, SICHRE of ROM C.

Construction of a 128 byte file-header under Amsdos and FutureOS

AMSDOS:

```
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00 UU N0 N1 N2 N3 N4 N5 N6 N7 E0 E1 E2 00 00 00 00 - User, Name, Ext.
10 00 00 TY 00 00 SL SH 00 LL LH AL AH 00 00 00 00 - Typ,St.,Len,A.St
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - unused
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 - unused
40 LL LH 00 CL CH ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? - Lenght, Checksum
50 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? - not defined
60 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? - not defined
70 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? - not defined
```

Abbreviations: Typ = file-type, St = Start-/Load-Address, Len = file-length and A.St = Auto-Start-Address. L = low, H = high.

Under FutureOS a file-header is expanded. The bytes which are defined by AMSDOS keep their function. But all other bytes are used too.

FutureOS:

```
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00 UU N0 N1 N2 N3 N4 N5 N6 N7 E0 E1 E2 I0 I1 I2 I3 - User, Name, Ext.
10 I4 I5 TT XX YY SL SH SB LL LH AL AH OL AB I6 I7 - Typ,X,Y,St,L,ASt
20 I8 I9 IA IB IC ID IE IF I0 I1 I2 I3 I4 I5 I6 I7 - Icon-data 08-17
30 I8 I9 IA IB IC ID IE IF I0 I1 I2 I3 I4 I5 I6 I7 - Icon-data 18-27
40 LL LH IT PL PH I8 I9 IA IB IC ID IE IF I0 I1 I2 - Len,IcoTyp,PrSm.
50 I3 I4 I5 I6 I7 I8 I9 IA IB IC ID IE IF I0 I1 I2 - Icon-data 33-42
60 I3 I4 I5 I6 I7 I8 I9 IA IB IC ID IE IF I0 I1 I2 - Icon-data 43-52
70 I3 I4 I5 I6 I7 I8 I9 IA IB IC ID IE IF I0 I1 I2 - Icon-data 53-62
```

FutureOS - 128 byte file-header, byte after byte:

```
Byte(s) ! meaning, function
--------!---------------------------
     00 ! Usernumber of file &00..&FE, don't use &E5 or &FF!
01 - 08 ! Filename (8 bytes)
09 - 0B ! Extent (3 bytes)
0C - 11 ! I00 to I05 (6 bytes) icon-data
     12 ! Filetype: &00 = basic, &02 = binary, &0A = Prowort
        ! Protected files: add &01, example: &01 = protected basic
        ! FutureOS files: add &80, 8. bit is set
     13 ! Horizontal range or X-width for semi-graphic-icon
     14 ! Vertical range or Y-lines for semi-graphic-icon
15 - 16 ! Low, High-byte start/load-address of file
     17 ! Physical RAM-block of start/load address: &C0,&C4,&C5-&FF
18 - 19 ! Low, High-byte file-length (first 16 bit)
1A - 1B ! Low, High-byte auto-start-address, else &0000
     1C ! Over-byte of file-length ==> file-length up to 16 MB
     1D ! Physical RAM-block of auto-start-address: &C0,&C4,&C5-&FF
1E - 3F ! I06 to I27 (34 bytes) icon-data
40 - 41 ! Repeat of L, H-byte file-length, but NOT USED!
     42 ! Icon-type: text-, graphic- or semi-graphic icon
43 - 44 ! Low-, High-byte 16 bit checksum of this file-header (bytes 0-66)
45 - 7F ! I28 to I62 (59 bytes) icon-data
--------!---------------------------------
```

Such a header contains &63 = 99 byte icon-data. The icon-type-byte (&42) decides which kind of icon the file-header contains. This can be a graphic icon with 4 * 3 or 6 * 2 chars, a semi-graphic-icon or a text-icon. There are four different icon-types. All icons must be shown in screen mode 1 with 32 chars and 32 lines.

If the icon is a text-icon (icon-type is &00), the text can give information about the contents of the file or whatever.

A semi-graphic-icon (type &02) contains text in its icon-data too.

These chars are displayed as an ASCII-rectangle. The header-bytes &13 and &14 contain the rectangles X and Y range. But X * Y must be < &64.

Text and semi-graphic icons can contain all 256 chars. All chars are displayed as chars and NOT as control codes. The difference is that the text of a text-icon is displayed as one string, whereas a semigraphic icon is a rectangle made of chars.

The true graphic icons contain only screen mode 1 graphic-data. They can have the format of 2 * 3 (type &01) or 3 * 2 (type &03) chars.

Therefore a graphic-icon uses only 4 * 3 * 8 = 96 bytes from the 99.

The remaining three bytes can be used freely.

Programs with header under FutureOS:

Look under 'program architecture' for information about the different types of programs for FutureOS.

If you want to start a program, you have to tag it in the DIR. Then click the RUN icon. Now the first tagged file will be started. This program must contain a file header which provides information where to load and start the program.

If you want to create a program bigger than 44 KB, you should design it as a E-RAM program. A little loader can start the program itself if wanted. This loader can load the big program into free E-RAM.

The 16 KB blocks of the E-RAM will be switched in between &4000 and &7FFF under FutureOS. A E-RAM program is loaded completely into E-RAM, therefore it can have a length of up to 4 MB. This depends finally on the amount of connected expansion RAM.

OS functions for E-RAM management

FutureOS provides a detailed memory management system for up to 4 MB or expansion RAM (E-RAM). RAM variables of the OS being used for memory management have been described in section III (see page 15). In addition FutureOS provides a variety of OS functions which allow an application to deal with E-RAM in an easy way.
The smallest supported chunk of memory are blocks of 16 KB. This size was choosen, because it corresponds to the smallest independant size of memory the hardware can handle at once. 16 KB seem to be much when the CPC only has 64 KB E-RAM, but if you have 4 MB E-RAM then there are already 256 blocks of 16 KB. Today, in the 21. century most CPCs own 512 KB E-RAM (a X-MEM is 30 Euros only!), in this case you got 32 independent blocks. Looking at it from this perspective 16 KB seem to be a fair compromise, especially since it corresponds to the features of the hardware.

There are three kinds of OS functions dealing with expansion RAM:

1) OS functions to show how much E-RAM is available / connected

- FESB : Calculates free expansion memory (first 512 KB E-RAM)

- TXR4M : Checks how much E-RAM is available (up to 4 MB)

- GTPRB : Compiles a table of free E-RAM blocks

2) OS functions to allocate or free E-RAM for applications

- EFER : Allocates 16 KB E-RAM for a program (first 512 KB E-RAM)

- FER7F : Frees 16 KB E-RAM block in XRAM_?? variables

- SSB0 : Saves lower 16 KB RAM (&0000-&3FFF) in the E-RAM
- RRB0 : Restores lower 16 KB RAM (&0000-&3FFF) from E-RAM

3) OS functions performing calculations dealing with E-RAM

- NXX_ERM: Select next 16 KB E-RAM (out of 8 MB) defined in (AKT_RAM)
- NXT_ERM: Select next 16 KB E-RAM (out of 8 MB) defined BC

- LXX_ERM: Select previous 16 KB E-RAM (8 MB) defined in (AKT_RAM)
- LST_ERM: Select previous 16 KB E-RAM (8 MB) defined in BC

- E2XRAM : Converts physical E-RAM (&C4 - &FF) to pointer to XRAM_?? variable (first 512 KB E-RAM)

- XR2ER : Converts a pointer to a E-RAM variable (XRAM_C4, C5..FF) to the physical E-RAM block (&C4, &C5..&FF)
- BJKG : Converts a pointer to a E-RAM variable (XRAM_C4, C5..FF) to the physical E-RAM block (&C4, &C5..&FF). Set B = &7F.

Print characters and strings to a printer

An operating system is only worth something if one can bring the fruits of someone's hard work to paper - this way or similar people did argue due to the last millennium. According to today's values that's an outdated point of view. Or maybe not?

However, FutureOS provides everything an application needs to print characters, strings or graphics to a connected printer. Text can be printed very easy using 7 bits only, therefore the CPC6128 got a 7 bit printer port. Whereas graphics can be printed more efficient with 8 bits. Therefore the 6128 Plus received an 8 bit printer interface. But also the CPC6128 can easily have an 8 bit printer port: Cut D7 at the printer port off from Ground and then connect D7 to pin 12 (WR data) of the 8255 PIO of the CPC. The 8. bit can be set and reset in the same way one would write data to the tape port. But the user or the application won't need to bother about it, the OS functions of FutureOS will take care:

Common OS functions dealing with printers:

- XW_DR : Waits until the printer is ready to take a character

OS functions for the 7 bit print mode:

- DRZ7 : Prints a 7 bit character on the printer

- DRS7 : Prints a string (7 bit characters) on the printer

OS functions for the 8 bit printer mode:

- DRZP8 : Prints a 8 bit character on printer - Only 6128 Plus
- DRZO8 : Prints a 8 bit character on printer - Only CPC 6128 + Patch

- DRSP8 : Prints a 8 bit string on printer - Only 6128 Plus!
- DRSO8 : Prints a 8 bit string on printer - Only CPC 6128 + Patch

The configuration bytes of FutureOS will tell if work can be done in 7 or 8 bit mode. If the 7. bit (means bit 6) of the RAM variable KF_SIO is cleared, then only the 7 bit mode can be used. But if it's set then the 8 bit mode can be used in addition.

Useful OS functions

Now let's take a short look at some interesting OS functions. For further information please take a look at the files 'API-?-EN.DOC' of the API documentation.

Load files

- LADE_N : Load a file which is defined by drive, user number, name and extension (disc or hard disc)
- LADEN : Loads the first tagged file from disc or hard disc
- TEILA/B : Load a part of a file

Save files

- SICHRE : Saves a file, defined through drive, user number, name and extension (disc or hard disc)
- TEISI/K : Save a part of a file

Work with File Headers

- SHED : Shows a FutureOS file header in mode 1 (32*32)
- LADAH : Loads only the file header of a file
- TST_HED : Calculates the checksum of a file header

Show Files and Screens

- TYSAZ : Types a file on the screen. Screen mode 2 is used
- ZEIBI/J : Display a picture on the screen. A regular 17 KB screen or a compressed OCP screen can be used. Screen MODE and format can be selected using cursor keys
- OCPC0 : Uncompress OCP *.SCR file and show it on the screen

Set Screen Format

- S64X32 : Set screen to 64 columns and 32 lines
- S68X30 : Set screen to 68 columns and 30 lines
- S80X25 : Set screen to 80 columns and 25 lines

Management of Real-Time-Clocks

Read:

- R_RTC : Read time and date from a connected Real-Time-Clock (main function)
- R_SFRT : Read time and date from SF-II real time clock (RTC)
- R_SF3RTC : Read time and date from SF-III RTC
- R_LS3RTC : Read time and date from LambdaSpeak III / FS RTC
- R_NRTC : Read time and date from the Nova expansion

Write:

- S_RTC : Date and Time will be written into any connected RTC (main function)
- S_SFRT : Set time and date for SF-II real time clock
- S_SF3RTC : Set time and date of the SYMBiFACE III RTC
- S_LS3RTC : Set time and date of the LambdaSpeak III or FS RTC
- S_NRTC : Set time and date of the Nova expansion card RTC

Mathematics

- MUL88 : Quick 8 * 8 bit multiplication
- DIV88 ..: Quick 8 / 8 bit division

Management of mice and other HID devices

- R_ALM : Read positioning data from a connected proportional mouse
- R_PS2 : Read positioning data from PS/2 mouse of the SYMBiFACE II
- R_USB3 : Read positioning data from USB mouse of the SYMBiFACE III
- R_ALB : Read positioning data from USB mouse of the Albireo
- R_MPM : Read positioning data from MultiPlay mice
  (also see OS functions R_MPM1 and R_MPM2)
- G_GP2 : Get data from Hegetron Grafpad II

Conversion of bytes and data, binary and BCD

- CC2N : Converts two ASCII characters to one 8 bit value
- N_2_2C : Converts a byte to two ASCII characters
- HAUT : Displays one hexadecimal 8 bit value on screen
- BIN2BCD : A binary number (0-99) will be converted into a BCD number
- BCD2BIN : A BCD number will be converted into a binary number (0-99)

Copy, Fill and Clear memory areas

- F_FILL8 and F_FILL6 : the fastest possibility to fill CPC-RAM with an 8 or 16 bit value.
  Every byte is written in only 2 µs
- F_MOVE : A fast way to move memory blocks. Every byte is moved in only 5 µs
- LESC …….: The fastest way to clear the screen (&C000-&FFFF) like CLS

Debugging

- F_DUMP : Displays a part of the memory on screen, a DUMP
- EDIT : A memory area of &01E0 Bytes (actually being displayed) will be edited using the Hexadecimal mode
- F_PORT : Display and Edit all Ports of the CPC
- CARET : For a program this is the entry into the machine monitor, all Z80 registers are saved in its RAM registers

Sound-Management

- MAUS : Switch all sounds of the PSG off
- S_PSG : Writes data to the PSG

Check for connected hardware

- T_SF : Test if a SYMBiFACE II is connected
- T_SF3 : Test if a SYMBiFACE III is connected
- W_SF3 : Wait until the SYMBiFACE III does acknowledge to be ready

Return to FutureOS

- TUR_E : This is THE standard Jump into FutureOS for a program. Use it!
- TUR_D : Like TUR_E, but clears directory buffers and RAM variables TURBO_A to TURBO_M are initialized. Wallpaper will be reset
- KLICK : Jump back to FutureOS, if the upper part of the screen (icons!) isn't changed. Mostly used by little programs that alters only the lower half of the Desktop

Other OS functions

- FMD32 : Searches the first tagged file of all read DIRs

- GET_DIR : Reads DIRectories of all tagged drives

- LTERM_2 : Outputs a text on screen and via LamdaSpeak speech synthesizer

VII. Appendix

Icons

The Turbo-Desk uses icons extensively. All icons and icon sets (used to display time & date) need about 9,5 KB in ROM D. The addresses of the graphic-data of all icons and icon sets are listed in the file '**#EQU-API.ENG'**.

Every icon was painted to use it under screen MODE 2. It is 6 mode 2 chars broad and 3 chars high (6 * 3). Every pixel of the icon border must be set. If you want to include own icons or just change an icon you can do this by changing the data in ROM D, but ... attention!

An icon is included in ROM D in the following way: The first six bytes consist out of the graphic data from left to right, then come 6 bytes for the next scan-line below. But the most upper and most lower lines are missing, because they're always drawn as a line (all bytes &FF). This is made by ICON6ON (ROM D). Every icon consists therefore out of 22 lines (number of scan-lines - 2 = 8*3-2 = 22) with 6 bytes each.

If you have painted an own icon and want to install it into FutureOS just send it to me, I can install it for you.

Icon sets

The difference between icons and icons of numbers lie in their width. Icons from icon-charsets are only 3 (not 6) bytes broad. In ROM D they are saved like normal icons. First the most left byte of the first scan-line, then the middle-byte, then the right byte. Then the three bytes of the next scan-line below. And, like normal icons, they miss the upper and lower scan-line, because of these two lines are displayed as a line. To show such a number-icon use ICON3ON (ROM D).

Two icon charsets are existing in ROM D, they contain the numbers from zero up to nine. With ICON3ON you can display these big numbers in your own programs without problems. As long as you're working under screen mode 2 with 64 chars and 32 lines (like the desktop).

The mouse-arrow

The old generation CPCs use a software-sprite as mouse-arrow, whereas the 6128plus uses a hardware-sprite for this function. Hardware-sprite zero (memory-mapped address &4000) is used.

If you want to change the mouse-arrow you can create and install your own arrow into ROM D. Or let me install it for you.

The 6128plus version of FutureOS uses a compressed sprite format (128 bytes instead of 256), whereas the soft-sprite for the old generation CPCs isn't compressed.

Ways to control the mouse-arrow

To communicate with FutureOS (that means to move the mouse-arrow) you can use different instruments. The multiscan environment recognizes whatever you're using. The following possibilities are given:

- Joystick 0 or Joystick 1, with fire-buttons 0 and 1.
- Cursor-keys + Copy + little Enter.
- Joystick-compatible mouse or the Atari-ST mouse.
- MultiPlay Joystick 0 or 1
- PS/2 mouse of the SYMBiFACE II
- USB mouse of the SYMBiFACE III
- Marconi Trackball, Joy-compatible Trackballs or Atari-Trackball.
- Analog-Joystick (only 6128plus).
- LightPen + Copy (today only for 6128plus).

Differences between CPC old generation and 6128plus

FutureOS exists in two versions, one for the CPCs old generation and one for the Plus series CPCs. As programmer of the OS I kept the differences very little.

A FutureOS program should be able to run under both versions, because the differences don't influence the application / programming environment.

The FutureOS version for the CPCs is 100% compatible to the 6128plus.

The main-difference between both versions is the sprite (mouse-arrow) you use to control the Desktop. Whereas the CPCs use a software sprite, the 6128plus uses a hardware sprite (with 15 colors).

Only the FutureOS for the old CPCs is able to work with an Atari-ST mouse or trackball, the reason lies in the 6128plus hardware.

On the other hand only the FutureOS version for the 6128plus supports an analog-joystick and a light-pen, the reason lies once more in the hardware. The CPCs simply have no port for analog-joysticks.

There is one more difference. In the machine-monitor you can configure your system. That means switch the lower ROM or RAM on, and select the external RAM which is banked between &4000 and &7FFF. The 6128plus version allows you, in addition, to switch the memory mapped groups on or off (between &4000 and &7FFF). This allows you to deal direct with the new memory mapped hardware of the 6128plus.

Attention: There is an incompatibility! The CPC464 can't be used 100% with FutureOS, because of the fact that the software sprite leaves a smear on the screen. Thats all. The problem is that a CPC464 can't use the RAM configuration &C3 which is needed for the software sprite. All other features are still running on a CPC464 too. In addition FutureOS likes expansion RAM and 64 KB RAM aren't much. If you connect expansion RAM to your CPC464/664 which supplies your CPC with the corresponding banking mode &C3 (Revaldhinos 512 KB f.e.) the FutureOS should be able to run!

Class definitions of CPCs

Class 0 CPC: 64K RAM, no drive (CPC-464)

Class 1 CPC: 64K RAM, 3" drive (CPC-664)

Class 2 CPC: 128K RAM, 3" drive (CPC6128)

Class 3 CPC: 128K RAM, 3" drive, 2. drive: 80 tracks, doublesided

Class 4 CPC: 320K RAM, 3" drive, 2. drive: 80 tracks, doublesided

Class 5 CPC: 576K RAM, 3" drive, 2. drive: 80 tracks, doublesided

Class 6 CPC: 576K RAM, 3" & 2. drive (80 trks, DS), 20 MB hard-disc

Class 7 CPC: 2048K RAM, 4 internal & 4 external drives, 20 MB HD

Class 8 CPC: 4096K RAM, 2-8 drives, hard-disc

Only the space to read/write defines the CPC class. EPROM space (RO!) doesn't influence the CPC class.

FutureOS needs a class 2 CPC, it runs best with a class 4 CPC or higher.

The Hardware of the CPC

The I/O addresses of the Amstrad CPC computer:

The following list contains all port addresses (I/O) of the CPCs I was able to collect. This list may not be complete. The chars XX symbolize a byte with any possible content.

- &7FXX : Gate Array	write / -
- &BCXX : 6845 CRTC Address-Register	write / -
- &BDXX : 6845 CRTC Data-Register	write / -
- &BEXX : 6845 CRTC Status-Register	- / read
- &BFXX : 6845 CRTC Video-Address-Register	- / read
- &DFXX : select ROM	write / -
- &EFXX : Printer Port	write / -
- &F4XX : 8255 PIO Port A	write / read
- &F5XX : 8255 PIO Port B	- / read
- &F6XX : 8255 PIO Port C	write / -
- &F7XX : 8255 PIO Control-Register	write / -
- &F8B0 : Vidi-CPC Video-Digitiser - CRTC Index	write / -
- &F8B1 : Vidi-CPC Video-Digitiser - CRTC Data	write / -
- &F8E0 : Z80 STI Indirect Data Register	write / read
- &F8E1 : Z80 STI Gen. Purpose I/O Interrupt	write / read
- &F8E2 to & F8E7 : Interrupt, not usable	
- &F8E8 : Z80 STI Pointer Vector Register	write / read
- &F8E9 to & F8EB : Timer, not usable	
- &F8EC : Z80 STI USART Control Register	write / read
- &F8ED : Z80 STI Receiver Status Register	write / read
- &F8EE : Z80 STI Transmitter Status Register	write / read
- &F8EF : Z80 STI USART Data Register	write / read
- &F8E2 to &F8E4 : Dobbertin Eprommer 4003	? / ?
- &F8F2 : Dobbertin Eprommer 4003	? / ?
- &F9B0 : Vidi-CPC - Config(write), Capture(read)	write / read
- &F9FC to &F9FE : Otten & Fecht 1 MB RAM-Disc	? / ?
- &FA7E : Floppy Motor Control	write / -
- &FADC : Z80-SIO / DART port A Data Register	write / read
- &FADD : Z80-SIO / DART port A Control Reg.	write / read
- &FADE : Z80-SIO / DART port B Data Register	write / read
- &FADF : Z80-SIO / DART port B Control Reg.	write / read

- &FB7E : 765 FDC (internal) Status Register - / read
- &FB7F : 765 FDC (internal) Data Register write / read

- &FBDC : 8253 Timer counter 0 write / read
- &FBDD : 8253 Timer counter 1 write / read
- &FBDE : 8253 Timer counter 2 write / read
- &FBDF : 8253 Timer Modus Select write / -

- &FBE0 : Hard Disc Data Port write / read
- &FBE1 : Hard Disc Status, Reset write / read
- &FBE2 : Hard Disc Select, Configuration write / read
- &FBE3 : Hard Disc DMA, Interrupt write / read
- &FBE4 : Hard Disc Reset write / read

- &FBE0 to &FBE3 and &FBE8 : dk'tronics RTC. ? / ?

- &FBEE : SSA1 dk'tronics Speech-Module write / read

- &FBF0 to &FBFF : Otten & Fecht 1 MB RAM-Disc ? / ?

- &FBF6 : 765 FDC (Vortex, ext) Status Register - / read
- &FBF7 : 765 FDC (Vortex, ext) Data Register write / read

- &FD00 to &FD21 : CPC-IDE / Symbiface write / read

- &FEE8 and &FEEA : Multiface II ? / ?

- &FF00 to &FF2A : CPC-Booster(+) write / read

The most up to date version of this file can be downloaded from the internet.

FutureOS Homepage: http://www.FutureOS.de