File: API-D-EN.DOC (22 pages) February 25, year of the CPC 2019

Documentation of FutureOS functions located in ROM D

All OS functions are described in the following form:

1. Short description: describes an OS function in brief.

2. Label: this label labels the OS function in the (Z80-)source code. The actual label-library #EQU-API.ENG (delivered with the OS) contains the correct and newest address for every OS function. Please use EVER the appropriate LABEL when using an OS function or system-variable! Never use the direct address! addresses can change in further versions of FutureOS, but the features of OS functions and system-variables should stay through future.

3. ROM-number: gives the logical number of the ROM in which the appropriate OS function is located. More numbers are possible, if the function is present in different ROMs. This logical ROM number doesn't have to be equivalent with the physical number of the ROM. Only older versions of FutureOS have identical logical and physical ROM numbers (&0A, &0B, &0C and &0D). Now ROM expansions are bigger.

4. Start address: gives you the entry address of the current OS function in it's ROM. If the OS function exists in more than one ROM there may be more addresses given. This address is the same you'll find in the file #EQU-API.ENG for every label.

5. Jump in conditions: the conditions when the OS function get called are given here. Registers and RAM-variables must be load with the correct values. Which they are is described here.

6. Jump out conditions: describes registers and RAM-variables at the time when the OS function returns. Often registers give information if the OS function did return successful or not.

7. Manipulated: all the manipulations which was done through the OS function are given. Registers and RAM-variables are most interesting. But changes in memory or what ever are described, too.

8. Description: this is the complete description of the functions of the entire OS function.

9. Attention: crucial details are described here. All the OS functions are trimmed to high-speed. Therefore you have to deal with them in a correct way. If not something unmeant could happen.

BEWARE: Some OS functions of ROM D aren't very usefull for the programmer, beause they're used for the Desktop. High-Speed was the main target, so some of them run very static and have low resistance against wrong usage.

Since a variety of OS functions doesn't jump back it could be usefull to load the variables of the OK icon. This enables to jump back by using the OK icon of the Desktop.

The newest version of this file can be found in the internet:

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

DISPLAY A MODE 2 ICON 6*3 CHARS, USING 64*32 CHARS

Short description: One Mode 2 Icon of 6 * 3 chars will be displayed.

Label: ICON6ON

ROM-number: D

Start address: &C017

Jump in conditions:
DE = Target-address in the video RAM, normally between &C000 - &C7FA.
HL = Source-address of the icon data block (RAM or ROM D).

Jump out conditions: Icon was displayed.

Manipulated: AF, BC, DE, HL and 144 bytes video RAM.

Description: This OS function is used to display the square icons of FutureOS. Such icons are 6 Mode 2 chars in width and 3 lines in hight.

So the icon has 48 * 24 pixel (Mode 2). The screen must be set to screen Mode 2 and 64 chars * 32 lines, what is usual for the Desktop.

When calling ICON6ON the register DE must contain the target address inside the video RAM, normally located between &C000 and &C7FA.

Further the register HL must contain the source address of the icon data. But beware: The upper and lower raster line a every icon is displayed as a line. So this saves 12 bytes of icon data. Further all icons must have a frame. Some calculation time is also saved.

The icon data block contains the following data: First the six bytes of raster line two, from left to right. Then the six bytes of raster line 3, then line 4, and so on. At last the data block contains the six bytes of the 23. raster line (left to right).

Attention: Use the icon display OS function only in the 64 * 32 mode. The most upper and the most lower raster line (1. and 24.) of the icon is always displayed as white bar, its data is NOT located in the icon data block.

DISPLAY A MODE 2 ICON 3*3 CHARS, USING 64*32 CHARS

Short description: One Mode 2 Icon of 3 * 3 chars will be displayed.

Label: ICON3ON

ROM-number: D

Start address: &C05B

Jump in conditions:
DE = Target-address in the video RAM, normally between &C000 - &C7FA.
HL = Source-address of the icon data block (RAM or ROM D).

Jump out conditions: Icon was displayed.

Manipulated: AF, BC, DE, HL and 72 bytes video RAM.

Description: The OS function ICON3ON resembles ICON6ON (see before).

But this OS function displays icons of half width (three chars, not six). So the graphic data only contain three bytes for every scan line. The OS uses ICON3ON for example to display the numbers of time and date when using the Desktop.

SHOW TIME AND DATE ON DESKTOP

Short description: Displays the actual time and date on the screen. Or only a part of it.

Labels: DS_ZD (DS_YD, DS_XD, DS_WD, DS_VD or DS_UD)

ROM-number: D

Start addresses:

&E35C (DS_ZD)	///	&E39E (DS_YD)	///	&E3E0 (DS_XD)
&E422 (DS_WD)	///	&E464 (DS_VD)	///	&E4A6 (DS_UD)

Jump in conditions: The following RAM / system variables must contain correct values:

UHR_SEK	= Second .. &00-&59
UHR_MIN	= Minute .. &00-&59
UHR_STU	= Hour ..... &00-&23
UHR_TAG	= Day ..... &00-&31
UHR_MON	= Month ... &01-&12
UHR_JAR	= Year ..... &00-&99

Jump out condition: Time and date were displayed in 24*24 pixel iocns.

Manipulated: AF, BC, DE, HL, IX and the video RAM.

Description: The Desktop used this OS function(s) to display time and date on the screen.

Time and date (DS_ZD) are displayed on fixed screen coordinates. Icons of 24 * 24 pixel are used. That equals 3 * 3 Mode 2 chars for every number.

If you use another label than DS_ZD, then only a part of the time/date information is displayed. This makes sense if NOT all values of time and date have changed since they were displayed the last time. The labels display the following information:

DS_ZD	shows second, minute, hour, day, month and year.
DS_YD	shows second, minute, hour, day and month.
DS_XD	shows second, minute, hour and day.
DS_WD	shows second, minute and hour.
DS_VD	shows second und Minute dar.
DS_UD	shows second only.

Well, the programmer can hardly use this OS functions, because the target addresses of the time- and date-icons are fixed. One example could be that the Desktop should stay partly active when the interrupts are switched off.

Attention: The target addresses in the video RAM are fixed.

KLICK - ENTRY POINT TO DESKTOP WITH INTACT ICON-SET

Short description: Jumps back to the Desktop. Usable when upper half of the screnn (icons) is intact.

Label: KLICK

ROM-number: D

Start address: &FE9A

Jump in conditions: Screen must be set to 64 * 32 chars.
The icons in the upper half of the screen must be unchanged.

Jump out conditions: There is NO return.

Manipulated: - not relevant -

Description: KLICK is used when a program wants to turn back control to the Desktop. It should ONLY be used when the screen has mode 2 and is in 64 columns / 32 lines format. Further all icons of the upper screen half must be still intact. If this is not the case, you should better use the entry points TUR_D or TUR_E.

The following actions are done by entry point KLICK:
- Drive motors are switched on/off depending on system variable MO_ST
- System variables for mouse-pointer management were restaurated
- Standard 64 KB RAM is switched on
- then the control is given to the OS

Attention: There is NO return, because KLICK is not a OS function, it is a kind of enty point to the OS.

INITIALIZATION OF OS - MAIN ENTRY POINT TO OS

Short description: The two main entry points to the OS.

Labels: TUR_D (switches all drives off additionally) // TUR_E (normal)

ROM-number: D

Start addresses: &FEA0 (TUR_D) // &FE9D (TUR_E)

Jump in conditions: -

Jump out conditions: There is NO return.

Manipulated: look below...

Description: This two entry points are used to give control back to the OS after the (temporary) END of a program. They do the following:
- set screen to mode 2 and 32 lines of 64 columns.
- clear screen.
- show drive-, graphic-, time-/date- icons.
- jump to entry point KLICK (look before for further information).

The entry point TUR_D will additionally set all drives to NOT TAGGED, all previously buffered DIRectories are ignored. Sets system variable DIRIN to &FF. Further system variables TURBO_A..M were initialized by reading configuration from system variables INI_MED (look there).

Attention: There is NO return. When using TUR_D all buffered DIRectories will be set unvalid, all drives are set to NOT-TAGGED.

(UN-)TAG A FILE FOR USAGE

Short description: The file located under the mouse-pointer will be tagged for further usage.

Label: TAG_DIR

ROM-number: D

Start address: &FEB5

Jump in conditions: The system variables must contain correct values

Jump out conditions: NO return. OS function jumps to entry point KLICK

Manipulated: File-Tagging-Byte of the file under the mouse-pointer

Description: After reading and buffering the DIRectory of a drive under FutureOS the user must first tag the files which should be used.

TAG_DIR does this. It tags a untagged file or a previously tagged file will set back to untagged status. The file under the mouse-pointer is used, that depends on the position of the mouse-pointer and the DIRectory page which is currently displayed.

Attention: There is NO return (jumps to KLICK). Well, this OS function is not of great interest for the programmer.

SHOW UP TO 64 FILE-NAMES OF ONE DIRECTORY

Short description: Show pages of buffered DIRectories, browse through the pages of different buffered DIRs. Status line will be shown.

Labels: SCRI shows the first page of the actual buffered DIRs. SC_AU turns one page up and SC_AB turns one page down.

ROM-number: D

Start addresses: &FEAC (SCRI) // &FEB2 (SC_AU) // &FEAF (SC_AB)

Jump in conditions: The following RAM variables must be correct:

DIRIN	= &FF => no DIR, abort OS function.
TDRAM	= RAM block of 16er DIR of the actual shown drive.
TDHST	= HighByte of DIR in this RAM block: &40-&78.
TDANZ	= Number of 1K pages of the actual drive, max. 8 (HD)
TDAKT	= Actual 1K page of actual drive 0,1..3..7
TDLWK	= Actual drive, its DIR is displayed at the moment.

Jump out conditions: One 1K page (64 DIR entries) were shown on screen

Manipulated: - really much -

Description: You use this OS functions if you browse through the pages of the different buffered DIRectories in the Desktop.
SCRI shows the first 64 entries/file-names on the screen. If the DIR has less than 64 files, then only this number of files is shown.
You can use SC_AB to show the next page of file-names. And if you want to show the previous page, just use SC_AU.

The three OS functions show up to 64 file-names/entries, depending of the number of files in the actual DIRectory. The lower half of the screen ist used, 16 lines with 4 file-names each.

To show the file-names the so called 16er DIRs are used. The 16er DIR is a down calculated version of the 32er DIR (thats the buffered copy of the DIR of the disc itself), prepared just to show file-names.

All 32er and 16er DIRs are buffered in RAM, so you can browse very fast throug the DIRs.

All three OS functions additionally show a status line below the 64 entries. This status line informes about: format of the disc, free space in KB, used side of disc.

Attention: Before you call one of this three OS functions, it would be good it you test if any DIRectory has been read and buffered. You can take a look at system variable DIRIN, if it is &FF then no DIR has been read.
The screen position of the shown DIR - page is fixed. It starts in the 15. line and ends in the 31. line. The line below is used as status line.

DEACTIVATE DRIVE ON DISC-ERROR

Short description: Marks drive as not existing and deactivates it.

Label: LADNT

ROM-number: D

Start address: &FEA9

Jump in conditions: A = Number of drive to deactivate 0..7 (= A..H)

Jump out conditions: There is NO return, jumps to entry point KLICK

Manipulated: AF, BC, DE, HL, DIRIN = &FF and TURBO_? = &02

Description: If a drive has reported an error, then the user should be informed about that. Therefore you can use LADNT.

You have to provide the number of the drive that should be deactivated in register A (0..7 means A..H). Then call LADNT, which set the drive-tagging-byte (TURBO_?) of the corresponding drive to &02, that means drive is not connected and not tagged. Further the "inactive" form of the corresponding drive- or partition- icon will be displayed on the Desktop screen (fixed video RAM position). And the system variable DIRIN will be load with value &FF, so all buffered DIRs are no longer valid.

The OS function LADNT doesn't return, it jumps to the OS entry point KLICK (see before).

If the user wishes that LADNT returns, then the OK icon can be used to jump back to a program or whatever. You have to prepare the variables of the OK icon before.

LADNT is especially usable for programs which work together with the Desktop (the OK icon is the connection).

Attention: NO return! OS function jumps to KLICK! The inactive version of the corresponding drive-icon will be displayed on the screen.

MARK DIR-RAMs IN XRAM_?? VARIABLES

Short description: All expansion RAM blocks which contain DIRectory buffers will be registered / marked in the XRAM_?? system variables.

Label: MADRA

ROM-number: D

Start address: &E2FC

Jump in conditions: TURBO_A..TURBO_M and XRAM_C4..XRAM_FF correct.

Jump out conditions: DE = &0804
The variables XRAM_C4 .. XRAM_FF maybe are marked as DIRectory buffer.

Manipulated: AF, BC, DE, HL, BC', D', HL' and XRAM_C4..XRAM_FF

Description: If a program reads a DIRectory "with hand" and buffers it into the expansion RAM, this RAM has to be marked as DIR buffer RAM.

This is managed through the XRAM variables, which provide information about the usage of every expansion RAM.

Attention: The DIR expansion RAMs of all tagged drives will be calculated, so the variables TURBO_A..TURBO_M must contain correct values.

DISPLAY DRIVE AND OTHER ICONS

Short description: The icons of all drives/partitions and some other icons will be displayed on the screen.

Label: AMON

ROM-number: D

Start address: &E4E8

Jump in conditions: The screen must be set to screen mode 2 with 32 lines and 64 columns each. The RAM variables TURBO_A .. TURBO_H must contain correct values.

Jump out conditions: Icons have been displayed.

Manipulated: AF, BC, DE, HL and the video RAM.

Description: This OS function displays all drive icons from A up to M and some other icons at its correct position on the screen. This can be important if you want to restaurate the screen after usage.

Beside the drive / partition icons the following icons are also shown:

OK, REN, DRUCKEN, RETAG, UNTAG, WECKER, END, IDE and INFO.

This OS function AMON can display all icons of the Desktop (that normally can be seen), when used together with the OS function GRA_ICO or TXT_ICO.

Attention: The screen format should be set to mode 2, 64 columns and 32 lines.

INITIALIZE THE DRIVE SYSTEM VARIABLES USING THE CONFIG BYTES

Short description: TURBO_A, B, .. M will be set according the system configuration bytes.

Label: INI_MED

ROM-number: D

Start address: &E6AD

Jump in conditions: RAM configuration bytes must be correct.

Jump out conditions: TURBO_A, B, .. M initialized (using &00 or &02).

Manipulated: AF, B, DE, HL and TURBO_A, B, C, .. M

Description: This OS function is used to initialize the system variables TURBO_A, B, C - M. This makes sense -for example- after the alteration of the drive-bytes of the configuration bytes in RAM.

In the system variables TURBO_A..M the bit 1 is of special importance, because it defines if a drive / partition is connected or not. All other bits will be cleared to zero in every case. So every drive is defined as not tagged and side 0 is active.

After the call of INI_MED the system variables TURBO_A, B .. M contain the value &00 (drive is connected) or &02 (drive is NOT connected).

Attention: OS function reads the configuration bytes from ROM (NOT from the ROM). If the RAM configuration is changed, then the contained data will be realized.

SHOW THE TEXT STYLE ICONS

Short description: All text icons will be displayed on the Desktop.

Label: TXT_ICO - **OS FUNCTION IS SINCE FutureOS SYSTEM .8 HISTORICAL!**

ROM-number: D

Start address: &FEA6

Jump in conditions: Screen mode 2 in 64 * 32 format.

Jump out conditions: NO return, OS function jumps to KLICK.

Manipulated: AF, BC, DE, HL and the video RAM.

Description: Under FutureOS you can choose the icon style between graphic- or text- icons.

This OS function shows all text-style-icons of the Desktop, in fact that are the following icons: DIR, TYPE, LOAD, SAVE, ERA, COPY, MONITOR und RUN.

Attention: You should set system variable REG08_7 to value &01, just to tell the OS, that the text style icons are active.

THIS OS FUNCTION IS SINCE FutureOS SYSTEM .8 HISTORICAL!

SHOW THE GRAPHIC STYLE ICONS

Short description: All graphic icons will be displayed on the Desktop.

Label: GRA_ICO - **OS FUNCTION IS SINCE FutureOS SYSTEM .8 HISTORICAL!**

ROM-number: D

Start address: &FEA3

Jump in conditions: Screen mode 2 in 64 * 32 format.

Jump out conditions: NO return, OS function jumps to KLICK.

Manipulated: AF, BC, DE, HL, and REG08_7 is set to &00.

Description: Under FutureOS you can choose the icon style between graphic- or text- icons.

This OS function shows all graphic-style-icons of the Desktop, in fact that are the following icons: DIR, TYPE, LOAD, SAVE, ERA, COPY, MONITOR und RUN.

Attention: OS function sets system variable REG08_7 to value &00, just to indicate the OS that the graphic style icons are now active.

THIS OS FUNCTION IS SINCE FutureOS SYSTEM .8 HISTORICAL!

EXPAND DOBBERTIN-TIME(BCD) INTO SINGLE BYTES

Short description: Expand three Dobbertin compatible BCD encoded time bytes into six single nibble bytes.

Label: Z_D2Z

ROM-number: D

Start address: &FE88

Jump in conditions: The high-byte of the RAM address of all three source bytes and all six target bytes must be identical.
DE = Pointer to the first of six target bytes: H,H, M,M, S,S.
HL = Pointer to the last of three source bytes: Second, minute, hour
For example the system variable UHR_STU.

Jump out conditions: Register DE points to six target bytes in RAM.

Manipulated: AF, BC, E, L and the six target bytes.

Description: The Dobbertin real-time-clock (RTC) provides the time data in the so called BCD format: Seconds, minutes and hours are combined in one byte each.

Example: If the time bytes contain the values &56 (second), &37 (minute) and &18 (hour), then the actual time is 18:37:56 o'clock.

The Dobbertin time format is space sharing, but if you like to edit the time it is an advantage to access every number/nibble in one byte.
This OS function converts three Dobbertin / BCD compatible bytes (second -> minute -> hour, while HL points to the hour) into six single nibbles (hour high-nibble, hour lower-nibble, minute high, minute low, second high, second low). This six nibbles will be written to RAM beginning at the address in register DE, in corresponding succession. In the case of the abouve example: &01,&08,&03,&07,&05,&06 ab DE.

Beware: The pointer to the three source bytes (HL) points to the LAST of the three bytes. So HL points to the hour (BCD).
The generated time format of the six single time bytes is compatible to the OS function ZEEZ. ZEEZ allows to edit the time data video oriented. Look there...

Attention: The high bytes of the RAM addresses of all source bytes must be identical. It's the same for all target bytes. That means:
- DE < &XXFA
- HL > &XX02
when calling this OS function Z_D2Z.

Further the target bytes (beginning at DE) are writen upwards (that's normal), but the source bytes (ending at HL) are readed downwards in memory.

COMPRESS SINGLE BYTES TO DOBBERTIN-TIME (BCD)

Short description: Previously through Z_D2Z (look before) expanded bytes will be compressed again by this OS function into Dobbertin time format (BCD).

Label: Z_Z2D

ROM-number: D

Start address: &FE8B

Jump in conditions: The high-byte of the RAM address of all six source bytes and all three target bytes must be identical.
DE = Pointer to the first of six source bytes: H,H, M,M, S,S.
HL = Pointer to the last of three target bytes: Second, minute, hour
For example the system variable UHR_STU.

Jump out conditions: Under HL (from HL - 2 up to HL) three time bytes have been written to RAM.

Manipulated: AF, BC, E, L and the three target bytes.

Description: This OS function Z_Z2D is the counterpart to OS function Z_D2Z (look before). If you have previously expanded the Dobbertin / BCD time format into single nibbles, that means ZEEZ compatible format, then THIS OS function is used to compress the six time nibbles again into three BCD / Dobbertin time bytes. The six source bytes / nibbles will be read beginning at the address in register DE. And the three target bytes will be written into RAM at HL-2, HL-1 and HL.

In this case HL points again to the LAST of the three target bytes. In memory there are three bytes reserved for: Second, Minute and hour. HL has to point to the hour.

Attention: The high bytes of the RAM addresses of all source bytes must be identical. It's the same for all target bytes. That means:
- DE < &XXFA
- HL > &XX02
when calling this OS function Z_Z2D.

Further the source bytes (beginning at DE) are read upwards (that's normal), but the target bytes (ending at HL) are written downwards in memory.

EXPAND DOBBERTIN-DATE(BCD) INTO SINGLE BYTES

Short description: Expand three Dobbertin compatible BCD encoded date bytes into six single nibble bytes.

Label: Z_D2J

ROM-number: D

Start address: &FE8E

Jump in conditions: The high-byte of the RAM address of all three source bytes and of all six target bytes must be identical.
DE = Pointer to the first of six target bytes: D,D,M,M,Y,Y.
HL = Pointer to the first of three source bytes: Day, month, year.
For example the system variable UHR_TAG

Jump out conditions: Register DE points to six date bytes in RAM.

Manipulated: AF, BC, E, L and the six target bytes.

Description: The Dobbertin real-time-clock (RTC) provides the date data in the so called BCD format: Day, month and year are combined in one byte each.

Example: If the data bytes contain the values &21 (day), &04 (month) and &99 (year), then the actual date is march, 21 in 1999.

The Dobbertin date format is space sharing, but if you'd like to edit the date it's an advantage to access every number/nibble in one byte.

This OS function converts three Dobbertin / BCD compatible bytes (day, -> months -> year, HL points to day) into six single nibbles (day high-nibble, day lower-nibble, months high, months low, year high, year low). This six nibbles will be written to RAM beginning at the address in register DE, in corresponding succession. In the case of the abouve example: &02, &01, &00, &04, &09, &09.

The generated date format of the six single date bytes is compatible to the OS function ZEED. ZEED allows to edit the date data video oriented. Look there...

Attention: The high bytes of the RAM addresses of all source bytes must be identical. It's the same for all target bytes. That means:
- DE < &XXFA
- HL < &XXFD
when calling this OS function Z_D2J.

COMPRESS SINGLE BYTES TO DOBBERTIN-DATE (BCD)

Short description: Previously through Z_D2J (look before) expanded bytes will be compressed again by this OS function into Dobbertin date format (BCD).

Label: Z_J2D

ROM-number: D

Start address: &FE91

Jump in conditions: Das Highbyte aller sechs Quellbytes muº gleich sein, das Highbyte aller drei Zielbytes ebenfalls.
DE = Zeiger auf sechs Quellbytes: T,T,M,M,J,J.
HL = Zeiger auf drei Zielbytes: T, M, J. // z.B. UHR_TAG

Jump in conditions: The high-byte of the RAM address of all six source bytes and all three target bytes must be identical.
DE = Pointer to the first of six source bytes: H,H,M,M,S,S.
HL = Pointer to the last of three target bytes: Second, minute, hour
For example the system variable UHR_STU.

Jump out conditions: Beginning at register HL three date bytes have been written to RAM.

Manipulated: AF, BC, E, L and the three target bytes.

Description: This OS function Z_J2D is the counterpart to OS function Z_D2J (look before). If you have previously expanded the Dobbertin / BCD time format into single nibbles, that means ZEED compatible format, then THIS OS function is used to compress the six date nibbles again into three BCD / Dobbertin time bytes. The six source bytes / nibbles will be read beginning at the address in register DE. And the three target bytes will be written into RAM at HL.

Attention: The high bytes of the RAM addresses of all source bytes must be identical. It's the same for all target bytes. That means:
- DE < &XXFA
- HL < &XXFD
when calling this OS function Z_J2D.

EDIT DATE OR TIME VIDEO-ORIENTED

Short description: These two OS functions allow you to edit time (ZEEZ) or date (ZEED)video oriented.

Labels: ZEED (date) or ZEEZ (time).

ROM-number: D

Start addresses: &FE97 (ZEED) /// &FE94 (ZEEZ)

Jump in conditions: The system variable C_POS contains the upper left position (in screen mode 2 chars) where the date or the time should be edited. The field where the data shall be edited is three lines in hight and 22 columns in width. The screen must be set to mode 2 and to 64 columns * 32 lines format.
HL = Pointer to six bytes, that contain time or date; HL < &XXFA !!!

Jump out conditions: The Z flag of the Z80 informs about the success of the editing:
Zero flag cleared => All went fine, editing was ended through Copy.
Zero flag set => The editing was aborted by pressing the ESC key!

Manipulated: AF, BC, DE, HL, IX, C_POS, REG08_0,1, REG16_0,1,2 and the six bytes RAM beginning at HL.

Description: These two OS functions allow you to edit the time or the date of the OS video oriented. If you want to change the date then you will use OS function ZEED, to change the time use ZEEZ. Both OS functions are very similar.

Before you call one of this two OS functions you have to set the screen to mode 2 and to 32 lines with 64 columns. The system variable C_POS contains the most upper left position of the editing fiels. The editor fild is 22 columns in width and 3 lines in hight. Thats for using the two big sets of numbers. Further the Z80 register HL must contain a pointer to the actual date or time in expanded format.

Example:
Format of the date: &02,&01,&00,&04,&06,&09 => 21. 4.69
Format of the time: &01,&08,&03,&07,&05,&06 => 18:37:56
To generate such six bytes from the data provided by the Dobbertin RTC you can use OS function Z_D2J (date) or Z_D2Z (time). Look there.
The editing process is done by the cursor keys. Copy ends editing, while the ESC key aborts the editing process.
After the return of the OS function the zero flag reports in which way the editing process was ended. If the OS function was correctly ended with the COPY key the zero flag is cleared. Else if the editing was aborted throught the ESC key, the zero flag is set to one.
In every case the source bytes will have been changed. It can be an advantage to save them before elsewhere.

Attention: Before you call this OS function the screen must be set to mode 2. Further the screen needs 32 lines with 64 columns each. If one of the two OS functions returns with a zero flag set to one, then the user has aborted editing with the ESC key.
Before calling one of the two OS functions register L must be fewer than the value &FA !!! That means that the six source bytes must be located in the same 256 byte page. So HL and HL + 6 must have the same high byte.

RE-TAG ALL PREVIOUSLY TAGGED FILES OF ONE DRIVE / PARTITION

Short description: All previously used (previously tagged) files of one drive (shown streaked out) will be marked as tagged again.

Label: RTLW

ROM-number: D

Start address: &FE82

Jump in conditions: HL = Pointer to TURBO_A, B, C, .., M.
Jump out conditions: All used files are tagged again.

Manipulated: AF, BC', D', HL' and the TMS_? variable of used drive.

Description: If the Desktop has used some tagged files (they are shown underlined), then this files will be marked as "used" afterwards (now they're shown streaked out).
If you want to work with all "used" files again, then you should mark them all again as "tagged". This does this OS function. All "used" files of ONE drive/partition will be marked as "tagged" again. Other tagged files will not be influenzed, they remain tagged.
When calling RTLW the register HL must contain a pointer to the system variable TURBO_A, TURBO_B, .. OR .. TURBO_M, this depends on the drive or partition you want to use.
Application example:
First you select the drive through the usage of Z80 register A

```
LD   A,drive     ;Drive A..M, equals the value 0..12.
RLCA
RLCA
RLCA             ;multiplicate drive * 8
LD   HL,TURBO_A  ;Pointer to TURBO_A (first drive)
ADD  A,L
LD   L,A         ;TURBO_A + ( drive * 8 )

CALL RTLW  ;call OS function to re-tag all used files.
```

Before you call the OS function you can directly load the drive-tagging-byte from system variable TURBO_? and test if the selected drive or partition is active at all:

```
LD A,(HL)
```

Attention: All previously tagged files stay tagged. The selection of the drive is done through a pointer (HL) to one of the system variables TURBO_A..M.

UNTAG ALL FILES OF ONE DRIVE / PARTITION

Short description: All files of one drive or partition will be set not tagged status.

Label: UTLW

ROM-number: D

Start address: &FE85

Jump in conditions: HL = Pointer to TURBO_A, B, C, .., M.

Jump out conditions: All used files are set to "not tagged".

Manipulated: AF, BC', D', HL' and the TMS_? variable of the drive.
Description: All the files of one drive or partition A..M will be set to "NOT TAGGED" status. In the Desktop all tagged files are shown underlined, all used files were shown streaked out and all not tagged files are written normal.
When calling this OS function UTLW the Z80 register HL must point to one of the system variables TURBO_A, B, C, ..., TURBO_M. So the selection of the drive is done. Application example:
First you select the drive through the usage of Z80 register A

```
LD   A,drive     ;Drive A..M, equals the value 0..12.
RLCA
RLCA
RLCA             ;multiplicate drive * 8
LD   HL,TURBO_A  ;Pointer to TURBO_A (first drive)
ADD  A,L
LD   L,A         ;TURBO_A + ( drive * 8 )

CALL UTLW  ;OS function aufrufen, alle Dateien ent-markieren.
```

Before you call the OS function you can directly load the drive-tagging-byte from system variable TURBO_? and test if the selected drive or partition is active at all:

```
LD A,(HL)
```

Attention: The selection of the drive is done through a pointer (HL) to one of the system variables TURBO_A..M.

TEST IF HEGOTRON GRAFPAD 2 IS CONNECTED

Short description: Test if the Hegotron Grafpad 2 is connected.

Label: T_GP2

ROM-number: D

Start address: &E714

Jump in conditions: -

Jump out conditions: The Z80 register A reports if the Hegetron Grafpad 2 is connected to the CPC or not.
A = &00 and zero flag is set ------> NO GP2 connected.
A = &01-05 and zero flag cleared --> the GP2 IS connected.

Manipulated: AF and BC.

Description: This OS function tests if the Grafpad 2 from Hegotron is connected to the CPC. You call the OS function without any parameters.
After the return of T_GP2 the Z80 register A and the zero flag report about the existence of the Grafpad 2.
If the OS function returns with the value &00 in register A and with a set zero flag, then the Grafpad 2 can't be found. Else if the OS function returns with A unqual to &00 and with a cleared zero flag, then the Grafpad 2 is connected.

Example:

```
 CALL T_GP2     ;test if Grafpad 2 is connected.
 JR   Z,NO_GP2  ;NO GP 2 connected!

GP2 ... ;the GP2 IS connected!

...
...
```

Attention: -

SCANNING OF THE HEGOTRON GRAFPAD 2

Short description: Scanning of the X-, the Y-coordinates and both buttons (E and S) of the Hegotron Grafpad 2.

Label: G_GP2

ROM-number: D

Start address: &E725

Jump in conditions: -

Jump out conditions: The registers DE and HL report about the X- and the Y-coordinates and the buttons E and S.
DE = Y-coordinate, theoretical: &0000-&04FF, in real: &001B-&0484
HL = X-coordinate, theoretical: &0000-&05FF, in real: &0020-&05DF
And register H contains the state of the buttons E (bit 7, MSB of H) and S (bit 6 of H). For both buttons is valid, that a cleared bit reports a pressed key. If a bit is set, the key is NOT pressed.

Manipulated: AF, BC, DE, HL and BC'

Description: If the CPC is connected to the Hegotron Grafpad 2 (you MUST test this before!), then you can use this OS function G_GP2 to scan the actual coordinates of the graphic table pen and the status of the buttons. When the OS function returns DE contains the Y-coordinate of the pen. Theoretically that can be a value between &0000 and &04FF.
But in reality the graphic tablet covers values between &001B and &0484. (There can be differences between different units). Little Y values are located at the bottom, near the user. Big Y values are located upwards, away from the user. The register HL reports about the X coordinate (bits 0-10) and the status of the buttons. The X coordinate theoretical spans over values from &0000 up to &05FF. In reality the values lie between &0020 and &05DF. Little X values are located left, big X values are located right on the graphic tablet.
Further the most upper bits of register H (7 and 6) contain the status of both buttons of the Hegotron GP2.

H encodes the buttons E (bit 7) and S (bit 6):

- If bit 7 is cleared to 0, the key E (Entry, Exit) IS pressed now.
  If bit 7 is set to 1, the key E is actually NOT pressed.
- If bit 6 is cleared to 0, the key S (Select) IS pressed now.
  If bit 6 is set to 1, the key S is actually NOT pressed.

Example:

```
 CALL G_GP2 ;get Grafpad 2 coordinates and buttons.

 LD   (GP2_X_Koordinate),DE ;save X-coordinate into RAM

 LD   A,H
 AND  A,&C0             ;isolate bits 7 and 6 (= keys E and S),
 LD   (KEYS_E_AND_S),A) ;and save key status (??00 0000) to RAM

 RES  7,H
 RES  6,H
 LD   (GP2_Y_Koordinate),HL ;save Y-coordinate into RAM
...
```

Attention: Use this OS function ONLY if you are sure that the Grafpad 2 IS connected to the CPC. Use T_GP2 to test this (look before).

SCANNING OF THE BUTTONS E AND S OF THE HEGOTRON GRAFPAD 2

Short description: Keys E and S of the Hegotron Grafpad 2 are scanned.

Label: K_GP2

ROM-number: D

Start address: &E750

Jump in conditions: -

Jump out conditions: The register A reports if one of the two buttons E (Entry, Exit) and / or S (Select) is pressed.
Register A bit 7 corresponds to key E.
Register A bit 6 corresponds to key S.
A cleared bit (0) symbolizes a pressed key, whereas a set bit (1) symbolizes that the key is NOT pressed.

Manipulated: AF and BC.

Description: If the CPC is connected to the Hegotron Grafpad 2 (you must test this before!), you can use this OS function K_GP2 to investigate the status of both keys.
After the return of the OS function the status of the keys is encoded in the two upper bits of register A; key E (bit 7) and key S (bit 6):

- If bit 7 is cleared to 0, the key E (Entry, Exit) IS pressed now.
  If bit 7 is set to 1, the key E is actually NOT pressed.
- If bit 6 is cleared to 0, the key S (Select) IS pressed now.
  If bit 6 is set to 1, the key S is actually NOT pressed.

Example:

```
CALL K_GP2 ;scan Grafpad 2 keys

AND  A,&C0             ;isloate bits 7 and 6 (??00 0000)
LD   (KEYS_E_AND_S),A) ;and save status of keys to RAM

...
```

Attention: Use this OS function ONLY if you are sure that the Grafpad 2 IS connected to the CPC. Use T_GP2 to test this (look before).

READ DIRECTORIES OF ALL TAGGED DRIVES AND PARTITIONS

Short description: The directories of all tagged floppy disc drives and hard disc partitions will be read and buffered in E-RAM.

Label: GET_DIR

ROM-nummer: D

Start address: &FE7C

Jump in conditions: All floppy disc drives and hard disc partitions whose directories shall be read must be tagged active in the OS system variables TUBBO_A to TURBO_M (see file #OS-VAR.ENG).
There must be enough free expansion RAM (E-RAM) to be able to buffer all directories.

Jump out conditions: The directories have been read and buffered.
The accumulator (A) provides data about the success of the OS function

A = &00	=>	All directories have been read successfully.
A = &01	=>	No mass media was tagged, so nothing could be read.
A = &02	=>	There was not enough free E-RAM, nothing was read.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, TURBO_A..M
FDC, drive motors, lower RAM/ROM state, and some E-RAMs.

Description: This high level function of the FutureOS allows to read directories from floppy disc drives and the Dobbertin HD20 hard disc partitions and buffer them in the E-RAM.
The read directories will be sorted. Then a text copy of them (the so called 16er DIR) will be created, which only contains the file names.
This way it can be directly displayed on the screen (see OS functions SCRI, SC_AU, SC_AB in ROM D). To select drives or partitions for reading their directory you have to set bit 0 of the corresponding RAM variable (TURBO_A, B, C,... M). For all other devices you have to reset the bit 0 for not reading the directory. After returning from the OS function GET_DIR the accumulator (A) will tell you about its success. If A is cleared to &00 then everything worked just fine. But if A is set to &01, then there was no tagged device to read from. And if the A register is set to &02, then there was not enough E-RAM available to perform the function. If A is not zero, nothing was read.

This is a raw rule for expansion memory consumption:
Data, System format need a maximum of: 3 KB
Vortex format needs a maximum of.....: 6 KB
HD20 partition needs a maximum of.....: 24 KB
RAM disc (M) needs a maximum of.....: 2 KB
In addition a 16 KB E-RAM Block will be needed as buffer for lower RAM.
Example: Drive A (Data) and B (Vortex) need 3 + 6 + 16 = 25 KB E-RAM.

Attention: This high level OS function alters all Z80 registers except IY and I. It will not work if there isn't enough E-RAM.
Devices which are not wanted to be read must be untagged in their RAM variable TURBO_? (? = A..M).
This OS function is not part of older/previous FutureOS versions!

CONVERT TWO ASCII CHARS TO ONE 8 BIT VALUE

Short description: Converts two ASCII chars into an 8 bit value.

Label: CC2N

ROM-number: D

Start address: &FE7F

Jump in conditions: HL contains two ASCII characters (0-9 and A-F).
Register H holds the "Highbyte" and register L holds the "Lowbyte".

Jump back conditions: A = 8 bit value &00-&FF

Manipulated: AF and B.

Description: OS function CC2N is used to convert two ASCII characters into one 8 bit value. The two ASCII characters are given in registers H (High byte) and L (low byte). When calling CC2N the register HL must contain only correct characters: "0"-"9" or "A"-"F".

After the return of this OS function register A will contain an 8 bit.

Example:

The register HL contains &3742. Where &42 = ASC("B")".

```
LD   HL,&3742
CALL CC2N
```

Now register A contains &7B.

This OS function is only able to convert chars from "0" up to "9" and from "A" up to "F" into one 8 bit value.

Attention: Valid ASCII characters - that can be converted by CC2N - are "0" to "9" (&30-&39) and "A" to "F" (&41-&46).