Documentation of FutureOS functions located in ROM A

All OS functions located in "ROM A" are merely concerned with keyboard management, text, screen, printer, PSG and management of the Dobbertin HD20 hard disc.

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 the appropriate LABEL every time when using an OS function or system-variable! Never use the direct address, just for safety.

3. ROM-number: Provides the logical number of the ROM in which the appropriate OS function is located. Multiple numbers are possible, if the function is part of more than one ROM. The logical ROM number is probably not the same as the physical number of the ROM. The logical ROM numbers of FutureOS are &0A, &0B, &0C and &0D.

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

5. Jump in conditions: The conditions when the OS function gets called are given here. Registers and system RAM-variables must be loaded 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 flags and registers provide information about the success of the OS function.

7. Manipulated: All the manipulations which have been done through the OS function are listed. They can be in Z80 Registers, system RAM-variables, memory, I/O space or elsewhere.

8. Description: This is the complete description of all relevant details 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.

In this ROM you find the keyboard manager, the text manager, parts of the hard-disc manager, the printer manager. Functions for the CPC-IDE, X-MASS, SYMBiFACE II and III, the CPC-Booster and more. Also the DUMP and F-PORT Systems are in ROM A.

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

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

TEST AND READ STATUS OF ALL KEYS

Short description: This OS function senses the status of the keyboard. When a key is pressed, the corresponding ASCII value is returned. The status of SHIFT and CONTROL is noticed.

Label: H_ALLET

ROM-number: A

Start address: &C02F

Jump in conditions: Key-Translation-Tables for Normal, Shift, Control and the Shift+Control level must be intact. That means: After TAST_N, TAST_S, TAST_C and TAST_SC must be 80 Bytes for each level. They can contain values between &00 and &FE. But &FF or &00 should not be used!

Jump out conditions: A = ASCII value, if corresponding key is pressed
                             A = &FF if NO key is pressed

Manipulated: AF, BC, DE, HL and XL (= IX lower 8 Bits),
                 PIO: status, Port A and C. PSG: register 14, Latch address register.

Description: This function investigates if a key is pressed. If no key is pressed the accumulator = Z80 register A will be set to the value &FF. But if any key is pressed at the moment the Shift and the Control status it senses additionally.

You can distinguish four different states:
- neither Shift nor Control is pressed,
- Shift is pressed,
- Control is pressed or ...
- Shift and Control are pressed both.

Depending on the state of Shift and Control keys, the function reads from one of four tables. The keyboard is organized in a 10 * 8 Matrix. Therefore every table contains 80 Bytes. Depending on the state of Shift, Control and the pressed key, one table will be chosen and the corresponding ASCII value is returned in the A register (look file #EQU-API.ENG).

Example: Read keyboard until a key is pressed

```
 LD  BC,(&FF01)   ; C = physical number of ROM A, B=&DF
 OUT (C),C        ; select ROM A (from &C000 to &FFFF)

LOOP CALL H_ALLET ; read pressed key

 INC A       ; Test if A = &FF
 JR  Z,LOOP ; continue until a key is pressed
 DEC A       ; compensation of the INC A command
```

The A register (accu) contains now the ASCII value of the pressed key.

Attention: If you assign the value &FF to a key, you can't distinguish if no key is pressed or if the key with value &FF is pressed.

Remember: If no key is pressed this OS function returns with the value &FF in the accumulator.

TEST, READ STATUS OF ALL KEYS (TAKE CARE ABOUT THE CAPS LEVEL)

Short description: This OS function senses the status of all keys. If a key is pressed, the corresponding ASCII value is returned. This function cares about the CAPS-LOCK status and the keys SHIFT + CONTROL.

Label: H_XALLET (expansion of H_ALLET look before).

ROM-number: A

Start address: &C115

Jump in conditions: Key-Translation-Tables for Normal, Shift, Control and the Shift+Control level must be intact. After TAST_N, TAST_S, TAST_C and TAST_SC must be 80 Bytes for each level. They can contain values between &00 and &FE. But &FF is not allowed to be used!

The RAM variable CAPS must contain a valid value.

Jump out conditions:
A = ASCII value, if corresponding key is pressed.
A = &00 and the Z flag is set if the CAPS-LOCK key was pressed.
A = &FF if NO key is pressed.

Manipulated: AF, BC, DE, HL and XL and the PIO: status, Port A and C.
Further the PSG: register 14, Latch address register.
The content of the RAM variable CAPS could be changed, if the CAPS key was pressed.

Description: This OS function is an expansion of the OS function H_ALLET. Please refer to H_ALLET in addition!

If a key is pressed its value is returned in CPU register A. If no key is pressed the accu will be loaded with &FF and the S flag is set 1.

If this OS function returns with 0 in A (Z flag set) the CAPS key was pressed and therewith the keyboard level was changed.

The following control functions are available:
CAPS alone => switch CAPITAL-LETTERS permanent on.
CAPS + SHIFT => switch SHIFT-level permanent on.
CAPS + CONTROL => switch CONTROL-level permanent on.
CAPS + CONTROL + SHIFT => switch CONTROL+SHIFT-level permanent on.

Pressing a "normal" key (not CAPS or CONTROL or SHIFT) will return its value (depending on the keyboard level) in the A register.

If you press SHIFT, CONTROL or both additional to the key, then this information will be XORed with the state of the CAPS status. SHIFT (with activated CAPS-LOCK) lead to lower case letters.

<u>Example:</u>

If CAPS-LOCK is active and you press the key "s", then A will become the value of the ASCII char "S". But if you press "s" and additional the SHIFT key, you will get "s" from the OS function. SHIFT XOR SHIFT lead to the normal (standard) keyboard level.

The string editor STED uses this function, you can use STED to have a closer look at H_XALLET. For example by editing a file name ;-)

Attention: If you assign the value &00 or &FF to a key, you can't any longer distinguish if no key is pressed or if the key with value &00, &FF or CAPS-LOCK is pressed.

TEST IF ANY KEY IS PRESSED

Short description: This function investigates if any key is pressed.

Label: TST_TS

ROM-number: A

Start address: &C5FA

Jump in conditions: -

Jump out conditions:
Z-Flag set => Keyboard is FREE, no key is pressed.
Z-Flag cleared => some key is pressed at the moment.

Manipulated: AF, BC, DE, L and PIO, PSG

Description: This OS function enables you to test if any key is pressed or not. It doesn't wait for a key-press, it only investigates the actual status of the keyboard. This OS function doesn't need any parameters.

TST_TS returns the status of the keyboard in the Z flag. If the Z flag is set (Z,1), then no key is pressed and no Joystick is used. But a cleared Z Flag (NZ,0) is a sign of a pressed key of the keyboard or a move of one of the Joysticks.

To investigate which key is pressed you can use H_ALLET for example.

Attention: This OS function doesn't wait until a key is pressed. It just returns the actual state of the keyboard.

GET VALUE FROM &0 TO &F, PUT IT ON THE CRT

Short description: This OS function waits until a key from 0 to 9, a key from a to f or the DEL key is pressed. The status of the keys Shift and Control is not regarded and has no influence. If a value between &0 and &F has been read (from the keyboard), in addition it will be displayed on the screen (screen should be Mode 2) as a char which corresponds to that value ("0".."F").

Label: A_F_0_9

ROM-number: A

Start address: &C404

Jump in conditions: The cursor position in the RAM variable C_POS must contain a valid value else the char will be written anywhere in RAM.

The screen mode should be 2, because the char is displayed on the screen with a Print-function for MODE 2.

Jump out conditions: A = 0, 1, 2, .., 9, &A, &B, .., &F if the corresponding key from 0..9 or a..f had been pressed. But if the DEL key was pressed the accumulator (register A) contains the value &FD.

Manipulated: AF, BC, DE, HL and XL.
PIO status, Port A and C. PSG register 14, Latch Address Register.

Description: This function scans the keyboard until DEL is pressed or a key of the group "0"..."9", "a"..."f". If DEL was pressed register A returns the result &FD. If a key between "0".."9" or between "a".."f" was pressed then this function returns the corresponding value between &00..&09 or &0A..&0F in A. In addition the corresponding char is shown on the screen at the actual cursor position. For example if you press the key 8 then register A becomes the value &08 and the char "8" is shown on the screen. If you press "c" then A is set to &0C and "C" is shown in capital.

This function is used to read hexadecimal values from the keyboard.

Furthermore the read values are displayed on the screen (that should be Mode 2).

Attention: A key must be pressed before this OS function returns. But if you don't press a key from "0" to "9" or from "A" to "F" or DEL, then this OS function will not return. It will wait for a pressed key.

The displayed char will be shown with MODE 2 print functions on the screen. Therefore you should use this function under screen mode 2.

WAIT UNTIL KEYBOARD IS FREE

Short description: Wait until no key is pressed.

Label: WART_TS

ROM-number: A

Start address: &C4DB

Jump in conditions: -

Jump out conditions: returns when no key is pressed any longer.

Manipulated: AF, BC, DE and L.
                PIO status, Port A and C. PSG register 14, latch address register.

Description: This function scans the complete keyboard, the joystick ports, the mouse port and the fire-buttons of the analogue joystick if any key is pressed. The OS function waits until no key is pressed, then it returns. You can use this OS function to be sure, that a key (and any other key) which was recently pressed is not pressed any longer.

Attention: If a key is pressed constantly this OS function will never return! There are some expansions for the joystick port which will set a key (normally 3rt fire signal) pressed. In this case you shouldn't work with this function, because a return is not possible until you disconnect that expansion. The Atari-mouse adapter (Computer Partner, Schneider Magazin) for example set fire 3 to pressed! If you use this mouse with FutureOS, then better don't use WART_TS.

WAIT UNTIL THE KEYBOARD IS FREE OR THE WAITING-TIME IS OVER

Short description: While a key is pressed this OS function waits a defined time delay then it will return. But it will return immediately if no key is pressed.

Label: XWART

ROM-number: A

Start address: &C671

Jump in conditions: You have to set the following RAM variables:
VZ_MOD = Waiting-Mode (0 = first WT / 1 = following WT), normal 0.
VZ_AG = Waiting-Time (WT) first reload value, from &0000 to &FFFF.
VZ_AA = WT first actual remaining time, from &0000 to &FFFF.
VZ_FG = WT second reload value, from &0000 to &FFFF.
VZ_FA = WT second actual remaining time, &0000 - &FFFF.

Jump out conditions: This OS function will return if no key is pressed or if a defined WT (waiting time) is over.

Manipulated: AF, BC, DE, HL, RAM variables VZ_MOD, VZ_AA, VZ_FA and the status of the PIO and the PSG.

Description: This OS function is a combination. The first function is to wait until no key is pressed any longer. The second function is to return after a defined maximum waiting time (WT).

There are two kinds of WTs. The first type it the "first WT", the second type is the "second WT".

Thereby the "first WT" is the time the function waits when it's called the first time while a key is pressed. The other case, the "second WT" is the time to wait when this function is called a second, third and so on... - but a key must currently still be pressed.

If this function is called and no key is pressed the WT is set back to the "first WT" and it returns immediately.

If, at any time, no key is pressed, this function returns immediately. Before calling XWART you have to set five RAM variables. Well, usually this is done by the OS. However you can alter them. The 8 bit variable VZ_MOD defines the WT-status. Normally it is set to &00, that means "first WT". If you don't want to use a special first WT, you just have to set a value greater than &00.

The 16 bit variables VZ_AG and VZ_AA (look before) can contain a value between &0000 and &FFFF. But they must be set to the SAME value. They contain the "first WT", the time you have to wait when a key is pressed the first time.

The 16 bit variables VZ_FG and VZ_FA must contain an equal value too, it can be &0000 up to &FFFF. They define the "second WT", the time you have to wait if a key is still pressed or pressed longer.

For normal human beings you should use the following values to load the five WT variables:
VZ_MOD = &00
VZ_AG = &1000
VZ_AA = &1000
VZ_FG = &0200
VZ_FA = &0200

Attention: All the VZ variables are set when FutureOS starts, but you can alter them it you want to speed up or slow down the reactivity of the keyboard.

TEST A KEY OR A COMBINATION OF KEYS

Short description: Fast way to test a key or a combination of keys of the keyboard matrix.

Label: HOLE1TS

ROM-number: A

Start address: &C561

Jump in conditions:
H = Number of keyboard matrix row from 0 to 9. H is NOT allowed to become a value greater than 63 = &3F.
L = Mask of the key or the combination of keys to be investigated. This mask covers 8 bit addressed through the matrix row (see H). Every bit which is set, symbolizes a key to be tested.

Jump out conditions: Register A contains the value &00 and the Zero flag is set if the previously given key (or combination of keys) is (are) pressed.

If A is unequal to &00 and the Zero Flag is cleared then the chosen key was not pressed (or the combination of keys was not pressed).

Manipulated: AF, BC, DE and H
            PIO status, Port A and C. PSG register 14, latch address register.

Description: This function allows to test a defined key very fast. The A register and the Zero flag give information if the key was pressed at this moment or not. If you want to use this function you must know the construction of the keyboard matrix of the CPC. The only data you give to HOLE1TS is the keyboard row (0-9), where the key is located (register H) and which of the 8 keys (L) of this row should be tested.

It is possible to set more than one bit in that 8 bit mask to test more keys. In this case you will investigate a combination of up to 8 keys. But you only get the message "key(s) pressed" (A=0, Z=1) if ALL keys of that combination are pressed.

Example: You want to test the key "S". Then you have to load H with the value &07 (matrix row 7), because key "S" is located in the 7th row of the keyboard matrix. The key "S" is symbolized through the 5th bit in row 7. Therefore (to set the 5th Bit) you have to load register L with the value &10 (in binary: 0001 0000). Now you can call this OS function and test key "S".
If you want to test the combination of keys "S" and "D" then you load register L simply with the value &30 (in binary 0011 0000). You set one bit for every key you wish to investigate.

Attention: If you want to test a combination of keys, all that keys must be located in the same row (1 out of 10) of the keyboard matrix.
You cannot test a combination of keys, which are part of different rows of the matrix.

READ COMPLETE STATE OF KEYBOARD MATRIX

Short description: This OS function reads the state of all 80 keys and saves them in 10 Bytes in the RAM.

Label: HOLETST

ROM-number: A

Start address: &C585

Jump in conditions: HL = start of 10 bytes data in RAM, where the key information should be stored.

Jump out conditions: The actual status of all 80 keys is stored in RAM beginning at the address which was before in HL. Attention: A pressed key is symbolized through a cleared bit. If a bit contains "1" then the corresponding key was NOT pressed.

Manipulated: AF, BC, DE and HL
                PIO status, port A and C. PSG register 14, latch address register.

Description: This function reads the complete keyboard matrix in the RAM. Since the matrix contains 10 rows (which 8 bits each), 10 bytes of RAM are used to store keyboard data. Every key has its own bit. At the address of HL row 0 will be saved, to HL+1 the second row (row 1) will be saved. This continues up to HL+9 where the last row (row 9) will be stored.

Attention: This OS function reads data direct from the hardware, this data is not computed in any way. Therefore a cleared bit symbolizes a pressed key. So, if a bit is SET the corresponding key is NOT pressed.

SCAN OF THE CURSOR KEYS AND THE COPY KEY

Short description: With this OS function the user can specifically test the CURSOR keys (arrow keys) and the COPY key.

Label: H_CURA

ROM-number: A

Start address: &C250

Jump in conditions: -

Jump out conditions: The status of the five keys is provided in the A register. The five bits have the following meaning:

A = (only bits 0..4 are relevant)

Bit 0: Arrow UP
Bit 1: Arrow RIGHT
Bit 2: Arrow DOWN
Bit 3: Arrow LEFT
Bit 4: COPY

Manipulated: AF, BC, DE, L and PIO, PSG

Description: This function allows the scan of all four CURSOR keys and the COPY key. No parameters must be given when calling this function, PIO and PSG are programmed in the appropriate way.

H_CURA provides the keyboard status in register A. Only the lower five bits (0..4) are relevant. These bits from 0 to 4 contain the following keys: UP, RIGHT, DOWN, LEFT and COPY. Where a CLEARED bit symbolizes a PRESSED key.

Attention: Beware, a Zero-bit symbolizes a pressed key. If a bit is SET then the corresponding key is NOT pressed.

SCAN OF THE CURSOR KEYS, COPY AND LITTLE ENTER

Short description: This function investigates the state of the CURSOR keys, COPY and LITTLE ENTER.

Label: H_CCE

ROM-number: A

Start address: &C284

Jump in conditions: -

Jump out conditions: A = status of keys (bits 0..5 are relevant)
Bit 0: Arrow UP
Bit 1: Arrow DOWN
Bit 2: Arrow LEFT
Bit 3: Arrow RIGHT
Bit 4: Arrow COPY
Bit 5: little ENTER

Manipulated: AF, BC, DE, L and PIO, PSG

Description: This OS function is used to investigate the status of the CURSOR keys, the COPY key and the little ENTER key. The result will be provided in register A. The bits from 0 up to 5 have the following meaning: UP, DOWN, LEFT, RIGHT, COPY and LITTLE ENTER. A cleared bit symbolizes a pressed key. This OS function is Joystick compatible, that means that the four Cursor keys represent the four directions in which you can move the joystick. The Copy key emulates Fire 0 and the little Enter key is used instead of Fire 1. This OS function (H_CCE) and the scan OS function for Joysticks (H_JOY) are therefore bit compatible.

Attention: Beware, a cleared bit in register A symbolizes a pressed key. If a key is not pressed then the corresponding bit is set.

SCAN AND CONVERT CURSOR KEYS, COPY AND ESC

Short description: This OS function reads the status of the cursor keys, COPY and the ESC key. If ESC is pressed a special program follows. Else a byte is returned in register A, which informs you about the keyboard status.

Label: CUR_CPY

ROM-number: A

Start address: &C6C0

Jump in conditions: REG16_1 (RAM variable) must contain the content of register SP. Alternatively REG16_1 must contain a value for SP, which will be loaded into SP when ESC is pressed.

Jump out conditions: Depending on the pressed key, register A will contain a value.
Cursor UP   => A = &F0.
Cursor DOWN => A = &F1.
Cursor LEFT   => A = &F2.
Cursor RIGHT => A = &F3.
Copy pressed => A = &A4.

When ESC is pressed the contend from RAM variable REG16_1 will be copied to register SP and register A is cleared.

Manipulated: AF, BC, DE and SP if ESC was pressed.

Description: This OS function scans the status of the four cursor keys, COPY and ESC. If one of the cursor keys or the copy key is pressed, then register A will be loaded with one of the following values:

Cursor up......: register A contains value &F0.
Cursor down.: register A contains value &F1.
Cursor left…..: register A contains value &F2.
Cursor right…: register A contains value &F3.
Copy pressed: register A contains value &A4.

But if ESC is pressed, something special will happen: First register A will be cleared. Then the content of RAM variable REG16_1 will be read and written into the Z80 stack pointer register SP. If you use a wrong value for REG16_1 this could crash (!!!) the system after the next RET command.

Well, the data transfer from REG16_1 to SP is not only dangerous, it's a mighty tool too. You can load the contents of SP to REG16_1 at a defined point (while your program is running). Now it doesn't matter how much CALL instructions will be made (and manipulate the stack). If you call this OS function CUR_CPY and press ESC at any time, the old value will be reloaded into SP. Then the RET is done!

That means that if you press ESC, this OS function returns at that point in the program where REG16_1 has been loaded with the content of register SP. Test this OS function careful.

Before saving SP to REG16_1 you must use at least the CALL instruction once.

Attention: Normally the RAM variable REG16_1 must contain the same value as register SP.
Or it must point to another stack element.
For only testing the cursor keys, copy and so on, better use H_CCE.
This OS function is used by the F_PORT high-level system OS function.

SCAN OF BOTH DIGITAL-JOYSTICKS

Short description: The actual status of both joysticks is sensed and provided in two registers.

Label: H_JOY

ROM-number: A

Start address: &C2CC

Jump in conditions: -

Jump out conditions: The status of Joystick 0 and 1 is provided in the registers L and H. Only bits 0..5 are relevant.
L = Joystick 0
H = Joystick 1

The bits have the following meaning, whereas a cleared bit symbolizes a pressed key:
Bit 0: Joy 0 or 1 UP
Bit 1: Joy 0 or 1 DOWN
Bit 2: Joy 0 or 1 LEFT
Bit 3: Joy 0 or 1 RIGHT
Bit 4: Joy 0 or 1 Fire 0
Bit 5: Joy 0 or 1 Fire 1
Bit 6: Joy 0 or 1 Fire 2 - CPC old generation only! AtariST mouse f.e.

Manipulated: AF, BC, DE, HL and PIO, PSG

Description: To investigate the status of the two digital joysticks this OS function is used. The status of joystick 0 is stored in register L and the status of the second joystick (1) is stored in register H. The bits of both registers contain the same directions. In each case only the lower six bits are relevant. The bits from 0 to 5 contain: UP, DOWN, LEFT, RIGHT, FIRE 0 and FIRE 1. In every case a PRESSED key is symbolized through a CLEARED bit. This OS function is bit compatible to OS function H_CCE (look there).

Attention: If a bit is SET, the corresponding key is NOT pressed. The third Fire key (7. bit) is only provided with the old generation CPCs.
But the third Fire key should not be used, because the Plus CPCs don't have it. Thank you ASIC! Well, if you own a CPC Plus you can add this third fire button. It's only a wire and easy to do. Wish ya luck ;-)

SCAN DIGITAL-JOYSTICKS, MULTIPLAY, CURSOR KEYS, COPY & little ENTER

Short description: Scan of both digital joysticks (CPC and MultiPlay) and the cursor keys, copy and little enter. Result is provided in A.

Label: H_JC

ROM-number: A

Start address: &FE6B

Jump in conditions: -

Jump out conditions: Register A contains the keyboard data of the first joystick, the second joystick (CPC or MultiPlay) or the cursor keys with copy and little enter. This OS function is bit compatible to H_CCE and H_JOY.

The bits contain the following data, where a cleared bit equals to a pressed key.
Bit 0: UP
Bit 1: DOWN
Bit 2: LEFT
Bit 3: RIGHT
Bit 4: FIRE 0
Bit 5: FIRE 1
Bit 6: FIRE 2 - Only for CPC6128, Atari ST mouse or MultiPlay

Manipulated: AF, BC, DE, HL and PIO, PSG

Description: By calling this OS function you can investigate the status of both digital joysticks of the CPC, the two digital joysticks of the MultiPlay expansioin and the status of the cursor keys + copy + enter. Therefore the user is able to work with joystick 0 or 1 (CPC or MultiPlay) or with the cursor keys, copy and little enter. All five input devices are equal. But they is a priority in sensing the devices: Joystick 0 has the highest priority, followed by joystick 1, then joysticks 0 and 1 of th MultiPlay. Eventually the cursor keys, copy and little enter have the lowest (last) priority in device scanning. The input status in provided in Z80 register A. The lower seven bits are relevant. Bits 0...6 contain the following information:

UP, DOWN, LEFT, RIGHT, FIRE 0, 1 and 2. Every CLEARED bit symbolizes a PRESSED key / activated input. This OS function H_JC is bit compatible to the OS functions H_CCE and H_JOY (look there).

Attention: If a bit is SET it symbolizes a key which is NOT pressed. The third Fire button (bit 6) is provided for all CPCs old generation, but not at the Plus series of computers, Amstrad saved two diodes.
Well, if you own a CPC Plus you can add this third fire button. It's only one diode for every joystick port and easy to do.

SCAN OF CONTROL AN SHIFT KEYS

Short description: The status of the keys CONTROL and SHIFT is sensed.

Label: H_CS

ROM-number: A

Start address: &C2F7

Jump in conditions: -

Jump out conditions:
A = row 2 of the keyboard matrix
Bit 5: SHIFT
Bit 7: CONTROL

Manipulated: AF, BC, DE, HL and PIO, PSG

Description: You call this OS function to investigate the status of the keys CONTROL and SHIFT. Register A will be loaded with row 2 of the keyboard matrix. The status of the SHIFT key is provided in bit 5.

And bit 7 (the high-bit) contains the status of the CONTROL key. All other bits (0-4 and 6) represent other keys (look at the keyboard matrix).

Attention:
CLEARED BIT => KEY PRESSED
      SET BIT => KEY NOT PRESSED

CHECK IF MultiPlay IS FREE

Short description: This OS function tests if the MultiPlay is free or if it's in use right now. Both ports are checked.

Label: TMPA

ROM-number: A

Start address: &D9E6

Jump in conditions: -

Jump out conditions: The Zero Flag tells about the status of both joysticks or the MultiPlay.
Z-Flag is set: Either there is no MultiPlay connected at all. Or right in this moment there is no input coming from one or both of the Joysticks.
Z-Flag reset: In this moment there is some input coming from one or both of the MultiPlay Joysticks.

Manipulated: AF and BC.

Description: This OS function allows to check the status of both of the MultiPlay Joysticks. First it reads BIT 3 of the configuraton variable KF_MED. If this bit is cleared then there is no MultiPlay connected an this function will return with the Zero-Flag set.
But if the MultiPlay is connected, then both Joysticks will be read and tested. If there is no input from neither joystick the TMPA will return also with the Zero-Flag set.
But if the Zero-Flag is cleared, then there is some kind of input from one or both joysticks.

Attention: If this function returns the Zero-Flag set, then one can't know if either the MultiPlay is not connected or just not in use right now.
There is no special OS function to read the MultiPlay joystick data, because one just needs to read its ports:

```
First Joystick: LD BC,&F990 ;Port of 1. Joystick
                IN A,(C)

Second Joystick: LD BC,&F991 ;Port of 2. Joystick
                 IN A,(C)
```

In this case if a bit is set, it represents a pressed key / direction. (Usually when reading data from the keyboard or the Joysticks of the CPC this is the opposite. In case of the CPC keyboard/joystick every cleared bit does represent a pressed key).

BIT 0: Up
BIT 1: Down
BIT 2: Left
BIT 3: Right
BIT 4: First Fire Button
BIT 5: Second Fire Button
BIT 6: Third Fire Button (called SPARE in the manual)
BIT 7: 0

COPY CONTROL-CODE-TABLES TO RAM

Short description: Copies the control code tables (for mode 1 and 2) from ROM to RAM.

Label: CSTI

ROM-number: A

Start address: &C71C

Jump in conditions: -

Jump out conditions: A 64 byte control-code-table is copied to &B800 (for Mode 1) and to &B900 (for Mode 2) in RAM. One for each mode.

Manipulated: BC, DE, HL and &B800-&B83F, &B900-&B93F (incl. last byte)

Description: The FutureOS has own PRINT-CHAR(S) OS functions for Mode 1 and for Mode 2. Every mode (1 and 2) has its own control-code-tables (look there). There are two bytes for every control code in RAM. These two bytes contain a vector to a OS function which corresponds to the control code. So it is easy to include own control codes, just by setting these two bytes (look at handbook).

The 16 bit addresses for mode 1 are located between &B800 and &B83F. And the RAM between &B900 and &B93F contain the 32 addresses for mode 2 control codes. Control codes are the first 32 chars (0-31). Every control code has its own 2 byte target-address. Therefore every screen mode (1 and 2) need 64 bytes for the control-code-jump-table.

Now, this OS function does nothing else than copying the predefined control code tables from ROM A into RAM. Therefore every control code gets its own special function. If you have changed the control code tables (for example to add a new function), you can use this OS function to restaurate the control code tables.

Attention: This OS function overwrites all changes in the two control code tables.

PRINT A SINGLE MODE 1 CHAR ON THE SCREEN - PEN 01

Short description: A single char will be printed on the screen at the actual cursor position (Mode 1). Before, the cursor position is increased by 2 (equals one Mode 1 char). Pen 01 is used.

Label: PRI0BB

ROM-number: A

Start address: &C7B3

Jump in conditions: The cursor position (->C_POS) must contain a valid value between &BFFE and &C7FE.
Register L contains the ASCII code (0-255).

Jump out conditions: The cursor position is increased by 2 (= one Mode 1 char), so you can print direct the next char on screen.
One Mode 1 char has 16 bits = 2 bytes = 8 pixels horizontally. So the cursor position is increased by 2 for every char.

Manipulated: AF, BC, DE, HL and the RAM variable C_POS.

Description: This OS function is used to print a single (Mode 1) char on the screen. Screen mode 1 should be activated. The char is shown in the color of Pen 01. If you have not changed the color palette the char is shown in blue color on the screen. The RAM variable C_POS determines where the char will be printed on the screen. Since one Mode 1 char is 16 bit long, the cursor position will be increased by 2 through this OS function. So you can print a string (char after char on the screen) on the screen by calling PRI0BB several times. If you increase the cursor position by 1 after printing a char on the screen, you can show a string with a half-space between the chars. A string is then of better readability.

Attention: The data of the charset begins at address &3800. At this address the character ROM is localized in the LOWER ROM of the CPC.
The graphic data of 256 chars is located between &3800 and &3FFF. If you want to print a char on the screen, you must be sure that there is a charset (at &3800). So you have to switch the lower ROM on or you can load your own, free definable, 2 KB charset at address &3800.
Further it is necessary to be sure that RAM variable C_POS contains a value between &BFFE and &C7FE. If C_POS is corrupted the graphic data will be written anywhere in the RAM. This can cause a system crash!!!

This kind of print-one-char OS function exists for pen 1,2 and 3 of Mode 1 and you can print chars on the screen in two colors. The left half has another color than the right half. The next page shows all OS functions for Mode 1:

PRINT A SINGLE CHAR - MODE 1 - 1 OR 2 COLORS

```
 Label  ! Address ! Pen left half of char ! Pen right half
=================================================================
        !         !                       !
 PRI0GB !  &C73B  !    Pen 10 (Yellow)    ! Pen 01 (Blue)
        !         !                       !
-----------------------------------------------------------------
        !         !                       !
 PRI0BB !  &C7B3  !    Pen 01 (Blue)      ! Pen 01 (Blue)
        !         !                       !
-----------------------------------------------------------------
        !         !                       !
 PRI0GG !  &C84B  !    Pen 10 (Yellow)    ! Pen 10 (Yellow)
        !         !                       !
-----------------------------------------------------------------
        !         !                       !
 PRI0RR !  &C8E3  !    Pen 11 (Red)       ! Pen 11 (Red)
        !         !                       !
-----------------------------------------------------------------
```

The values 01, 10 and 11 for the pens are binary.
Pen 01 means therefore Pen 1.
Pen 10 means therefore Pen 2.
Pen 11 means therefore Pen 3.

The charset is a region of 2 KB, located in the lower ROM beginning at &3800.

If you want to use your own charset just load it into the RAM at &3800 and switch the lower ROM off (lower RAM on).

PRINT A SINGLE MODE 2 CHAR ON THE SCREEN

Short description: One single char is printed on the screen at the actual cursor position (OS function for screen Mode 2). Before the print, the cursor position is increased by 1.

Label: PR_2

ROM-number: A

Start address: &C9BD

Jump in conditions: The cursor position C_POS must contain a value between &BFFF and &C7FF. Register L contains the ASCII char, which should be printed (0-255).

Jump out conditions: A single char was printed on the screen. C_POS, the cursor position is increased by 1.

Manipulated: AF, B, DE, HL and the RAM variable C_POS

Description: PR_2 prints a single char on the screen. This OS function is made for the high-resolution screen mode 2. Before the char is printed on the screen the cursor position is increased by 1. The char in printed on the screen at the new position.

Attention: In Mode 2 it is possible to print a character with screen attributes.

You can print a char on the screen with the following attributes:

```
 Label ! Address ! Attribute and explanation
-------+---------+-------------------------
       !         !
 PR_2  !  &C9BD  ! normal print OS function, nothing special
       !         !
 PR_2D !  &D016  ! chars are streaked out (4. line set to &FF)
       !         !
 PR_2I !  &D064  ! chars are inverted (Pen <=> Paper)
       !         !
 PR_2K !  &D0BA  ! chars shown in italics
       !         !
 PR_2U !  &D113  ! chars are underlined (8. line set to &FF)
       !         !
---------------------------------------------
```

PRINT A STRING - SCREEN MODE 1 (NO CONTROL CODES)

Short description: A string up to 65536 chars will be printed on the screen. All 256 chars are printed as chars and not as control codes. Pen 01 will be used.

Label: STR_BB

ROM-number: A

Start address: &CA1C

Jump in conditions: BC contains the number of chars + 1 to be printed on the screen.
HL contains the address of the first char of the string. The other chars follow in memory upwards.
The RAM variable C_POS, which contains the cursor position, must contain a valid value between &BFFE and &C7FE.
If no own charset is loaded at address &3800, the lower ROM must be switched on. The lower ROM contains graphical data for all 256 chars, beginning at address &3800.

Jump out conditions: The string was displayed on the screen, at the cursor position (C_POS) + 2.
The cursor position is increased by the number of chars to be printed.
(C_POS) shows to the char which was printed at last. So you can print directly other chars after the last string.

Manipulated: AF, BC, HL, AF', BC', DE', HL' and RAM variable C_POS.

Description: This OS function is used to display a string of chars on the screen. This string can be up to 64 KB long. All 256 chars will be printed as their graphical symbol. All control codes are printed as graphic characters. To define the appearance of the chars you can either switch on the lower ROM, then you use the ROM charset located at &3800. Or you can load a 2 KB charset at address &3800 and switch the lower RAM on (lower ROM off).

Well, you want to show a string of chars on the screen. You have to load the 16 bit register HL with the address of the first char of this string. All other bytes have to follow in RAM directly and increasing.

The number of bytes, which should be displayed on the screen, is given in the 16 bit register BC. One thing, the screen mode should be set to 1. Now you can call this OS function STR_BB. The string will be printed in the color of PEN 01, it will be Mode 1 chars.

Attention: If you have not loaded any RAM charset and the lower ROM is switched off, no chars will be written on the screen. Maybe something unknown will be displayed on the screen in this case, but this can NOT crash the system.

Use this OS function only when screen MODE 1 is switched on, you have to switch it on yourself!

The following page will show the different OS functions in some more detail.

This OS function uses Pen 1, but you can use Pen 2 oder Pen 3 too. Further you can print a char in two colors. 4 pixel on the left and 4 pixel on the right have one color respectively.

EXPLANATION OF THE DIFFERENT STRING OS FUNCTIONS

```
! Label  ! Addr. ! Pen left ! Pen right !  Pen  !
!--------+-------+----------+-----------!-------!
! STR_GB ! &CAC6 !  Pen 10  !  Pen 01   ! P.2+1 !
!--------+-------+----------+-----------!-------!
! STR_BB ! &CA1C !  Pen 01  !  Pen 01   ! Pen 1 !
!--------+-------+----------+-----------!-------!
! STR_GG ! &CB50 !  Pen 10  !  Pen 10   ! Pen 2 !
!--------+-------+----------+-----------!-------!
! STR_RR ! &CBFA !  Pen 11  !  Pen 11   ! Pen 3 !
!-------------------------------------------------!
```

The numbers of the Pens 01, 10 and 11 are binary. Therefore Pen 01 is Pen 1, Pen 10 is Pen 2 and Pen 11 is Pen 3 decimal. Showing the Pen in binary says how the graphic data is written in the RAM. Every Pen (1, 2 and 3) has two bits per Pixel: One bit (the left or the right) or two bits can be set to define the Pen (and with the Pen the color).

PRINT A MODE 2 STRING WITHOUT CONTROL CODES

Short description: A string of up to 65535 chars will be displayed on the screen (Mode 2). Control codes are shown as chars.

Label: STR_2

ROM-number: A

Start address: &CCE0

Jump in conditions: BC = number of bytes of the string + 1.
HL = start of the string in the memory (vector to first char).
The RAM variable C_POS, the cursor position, must contain a value between &BFFF and &C7FF.
There must be a charset beginning at &3800. Load a charset or switch the lower ROM on.
The screen mode 2 should be activated.

Jump out conditions: The cursor position is increased by the length of the screen (former in BC). The string is shown on the screen.

Manipulated: AF, BC, AF', BC', DE', HL' and RAM variable C_POS.

Description: This OS function prints a string (register HL contains the start address) on the screen. Screen mode must be set to 2. The length (loaded in register BC) of the string can be up to 64 KB. All the 256 chars (the control codes 0-31 too) are displayed as graphic symbols.

Attention: Before you call this OS function you should switch to screen Mode 2. The RAM variable C_POS must contain the right value.

And there must be a charset beginning at &3800 (in RAM or switch L-ROM on).

It is possible to print a string with the following attributes on the screen:

```
!-------------------------------------------------!
! Label  ! Screen attribute             ! Address !
!--------+------------------------------+---------!
! STR_2  ! normal, no attributes        !  &CCE0  !
!--------+------------------------------+---------!
! STR_2D ! chars are streaked out       !  &D161  !
!--------+------------------------------+---------!
! STR_2I ! chars are inverted           !  &D1C2  !
!--------+------------------------------+---------!
! STR_2K ! chars are italics            !  &D22A  !
!--------+------------------------------+---------!
! STR_2U ! chars are underlined         !  &D296  !
!-------------------------------------------------!
```

PRINT A STRING WITH CONTROL CODES IN MODE 1

Short description: A string of chars (maximal 64 KB) will be displayed with Pen 1 on the Mode 1 screen.

Label: TER_BB

ROM-number: A

Start address: &CD4C

Jump in conditions: HL = vector to the start address of the string which should be printed. The end of the string is defined through a null-byte (&00).

The cursor position (RAM variable C_POS) must contain a value between &BFFE and &C7FE. Further the lower ROM must be switched on, or a charset must be loaded at address &3800.

Jump out conditions: -

Manipulated: AF, DE, HL, AF', BC', DE', HL' and the cursor position.

Description: This OS function allows to print a string on the screen. The screen Mode 1 should be activated. Pen 1 is used. All control codes (0-31) were used. Every string is terminated through the byte &00.

If you call TER_BB the register HL should point to the first byte, the start of the string (which you want to print on the screen). Beginning at HL every byte is printed on the screen (char = 32..255) or the corresponding control code function will be activated (char = 0..31).

The chars between &00 and &1F (0-31) are control codes.

This OS function assumes that a charset is located at address &3800.

Attention: Before you call this OS function the cursor position (C_POS) must contain a value between &BFFE and &C7FE.

The programmer must care about the charset, which starts at &3800. You can either switch on the lower ROM or load a charset in the RAM.

You can print a string with control codes under Mode 1 with Pen 1, Pen 2, Pen 3 or a combination of Pen 1 and 2.

```
!-----------------------------------------------!
! Label  ! Addr. ! Pen left ! Pen right ! color !
!--------+-------+----------+-----------+-------!
! TER_GB ! &CDF6 !  Pen 10  !  Pen 01   ! 2 + 1 !
!--------+-------+----------+-----------+-------!
! TER_BB ! &CD4C !  Pen 01  !  Pen 01   ! Pen 1 !
!--------+-------+----------+-----------+-------!
! TER_GG ! &CE80 !  Pen 10  !  Pen 10   ! Pen 2 !
!--------+-------+----------+-----------+-------!
! TER_RR ! &CF2A !  Pen 11  !  Pen 11   ! Pen 3 !
!-----------------------------------------------!
```

Where the binary values 01, 10 and 11 represent Pen 1, 2 and 3.

PRINT A MODE 2 STRING WITH CONTROL CODES ON THE SCREEN

Short description: A string is printed on the screen (in Mode 2). All control code functions are used.

Label: TERM_2

ROM-number: A

Start address: &D48C

Jump in conditions: HL = first Byte of the Mode 2 string, which should be printed on the screen.

The cursor position (RAM variable C_POS) must contain a value between &BFFF and &C7FF. There must be a charset at address &3800.

Jump out conditions: -

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL' and C_POS, the cursor position.

Description: A string is displayed on the screen. The screen mode must be set to 2. All 32 control codes (0..31) are interpreted as control functions. When calling this OS function the 16 bit register must contain the address of the first byte of the string, which should be displayed on the screen.

Further there must be a valid charset beginning at the address &3800.

Alternatively you can switch the lower ROM on, which contains the system charset.

The string is terminated through a null byte. Than means that every byte &00 stops the output of the string on the screen.

Attention: You must care about RAM variable C_POS, the status of the lower ROM and the existence of a charset (at &3800).

You can display a string in Mode 2 with different attributes. Further look 'PRINT A MODE 2 CHAR':

```
 Label   ! Addr. ! Attribute of the string
---------+-------+------------------------------
 TERM_2  ! &D48C ! no further attributes
---------+-------+------------------------------
 TERM_2D ! &D2F7 ! all chars are streaked out
---------+-------+------------------------------
 TERM_2I ! &D358 ! all chars are inverted
---------+-------+------------------------------
 TERM_2K ! &D3C0 ! all chars are italics
---------+-------+------------------------------
 TERM_2U ! &D42C ! all chars are underlined
-------------------------------------------------
```

SET SCREEN TO 64 ROWS AND 32 LINES

Short description: The format of the screen will be set to 64 columns and 32 lines.

Label: S64X32

ROM-number: A and D

Start address: &D5A8 (logical ROM A) and &E6B5 (logical ROM D)

Jump in conditions: -

Jump out conditions: MAX_CRX, MAX_CRY will be set correct.

Manipulated: Flags, BC, DE, HL, CRTC register 1, 2, 4, 6 and 7.

Description: The format of the screen (in Mode 2) will be set to 64 columns (Mode 2 chars) per line and 32 lines per page. But neither the screen mode is changed not the screen is cleared.

SET SCREEN TO 64 ROWS AND 32 LINES

SET SCREEN TO 68 ROWS AND 30 LINES

Label: S68X30

ROM-number: A and D

Start address: &D5DB

Jump in conditions: -

Jump out conditions: MAX_CRX, MAX_CRY are corrected.

Manipulated: Flags, BC, DE, CRTC register 1, 2, 4, 6 and 7

Description: The format of the screen will be set to 68 Mode 2 columns and 32 lines.

SET SCREEN TO 80 ROWS AND 25 LINES

Short description: -

Label: S80X25

ROM-number: A and D

Start address: &D60E (logical ROM A), &E6E7 (logical ROM D)

Jump in conditions: -

Jump out conditions: MAX_CRX, MAX_CRY are corrected.

Manipulated: AF, BC, DE, CRTC register 1, 2, 4, 6 and 7.

Description: The format of the screen will be set to 80 Mode 2 columns and 25 lines. You know this screen format from Basic.

Attention: The screen mode will NOT be changed.

SET SCREEN TO 80 ROWS AND 25 LINES

DUMP OF THE MEMORY ON THE SCREEN IN MODE 2 AND IN FORMAT 68 * 30

Short description: 480 Bytes memory will be displayed on the screen in MODE 2, 68 columns and 30 lines. The output is provided in hexadecimal and in ASCII / FSCW.

Label: F_DUMP

ROM-number: A

Start address: &FE77

Jump in conditions: HL = start address of the memory dump. Beginning with this address 480 bytes will be dumped on the Mode 2 screen in hexadecimal and ASCII (FSCW).

Before you call this OS function you must set to screen to Mode 2 and to the 68 * 30 format.

Jump out conditions: The lower ROM is banked in (RAM/ROM status &7F82)

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL' and the RAM variables REG08_0, REG16_0 and C_POS. Further the 2 KB text screen memory has been used. And the lower ROM has been banked in (status &7F82).

Description: This OS function is used to dump 480 bytes of actually used memory on the screen.
The characters will be displayed in screen MODE 2, which must be set be the application before calling this OS function (by using code like LD BC,&7F82 : OUT (C),C or similar). In addition the application must switch the screen format to 68 columns per line and to 30 lines / page (using OS function S68X30). The 16 bit register HL must contain the start address of the 480 bytes memory block which should be dumped on the screen.
The memory will be displayed in that way: In every line you can see a 16 bit address (at the left side), 16 bytes in hexadecimal (middle) and the same 16 bytes in ASCII / FSCW (right end of the screen).
Therefore every line contains an address and the corresponding 16 bytes of memory (first in hexadecimal, then in ASCII).
This OS function is part of the machine monitor of the FutureOS.
When this OS function returns the lower ROM will be banked in!!!

Attention: The calling application must be calling from an address greater than &4000. Else this OS function would return to an address inside the lower ROM, which in turn will lead to a Crash!

Attention: This OS function uses the text screen memory. So all previously contained data will be lost.
Before calling this OS function the application must switch MODE 2 on and set the format of the screen to 68 columns and 30 lines.
This OS function can only be called from an address greater than &4000 in RAM, because before it returns it will bank in the lower ROM. Therefore a return to an address smaller than &4000 is impossible.

16 BIT INPUT AND OUTPUT OF ALL CPC PORT ADDRESSES

Short description: Display and edit all I/O port addresses of the CPC.

Label: F_PORT

ROM-number: A

Start address: &FE7A

Jump in conditions: The cursor-inverting sub-OS function CUR_INV must have been copied to RAM.

Jump out conditions: Port-addresses altered by the used have been sent to the corresponding I/O ports.
Both ROM's (lower and upper) have been switched on, the screen is set to Mode 2 and the 68 * 30 format.

Manipulated: AF, BC, DE, HL, IX, YH, AF', BC', DE', HL' and the RAM variables REG16_0, REG16_1 and C_POS.
The RAM/ROM status, the screen mode and the screen format have been changed (--> lower and upper ROM on, Mode 2, 68 * 30).

Description: This OS function is part of the machine monitor of FutureOS. Beware! Before you can use this OS function, you must copy the sub-OS function CUR_INV from ROM in the RAM. This OS function is used to send (/read) 8 bit values to (/from) 16 bit I/O addresses, that means to I/O ports of the CPC.
This OS function is controlled through the cursor keys and Copy. With the cursor keys you choose the I/O address. To read a value from an I/O address you just have to press COPY, then the input-value is shown immediately. To send a value to an I/O address just move the cursor to that address, then press COPY and enter the value (which should be sent to the I/O address). You have to enter a hexadecimal value (&00-&FF). Further it is possible to enter the high- and the low-byte of an arbitrary I/O address.
To quit this OS function just press enter (once or more) at any time.

Attention: A system crash may be the consequence of wrong input addresses or output values. Only a good coder should use this mighty system OS function. A novice could cause damage!
This OS function works with DISABLED interrupts, if necessary use the DI command before calling it.
It is a MUST to copy sub-OS function CUR_INV to RAM before calling F_PORT.

INPUT FUNCTION FOR 8 BIT VALUES DISPLAYED ON SCREEN

Short description: Input function for values between 0 and 255.

Label: RB_8 or RBB_8

ROM-number: A

Start address: &FE3B (RB_8) or &FE35 (RBB_8)

Jump in conditions: The cursor position C_POS (RB_8) or one position before the cursor position C_POS-1 (RBB_8) will be used to enter the value.

Jump out conditions: A contains the value entered by the user and the Carry flag is set to 1.

If the Carry flag is cleared (NC) the OS function was ended through the ESC key. Then please ignore the entered value.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY, PIO, PSG and the system RAM variable C_POS, which contains the cursor position.

Description: These two OS functions are used to enter 8 bit values (0-255) in MODE 2. The actual cursor position (OS function RB_8) or the position one to the left (OS function RBB_8) will be used. In the latter case the character left to the cursor will be overwritten, but only when using the decimal system. This way an possible "&" character for the entry of hexadezimal numbers can be overwritten automatically.
Both OS functions RB_8 and RBB_8 jump either to BIT8_IN or B8DIN, this depends on FutureOS using the hexadecimal or the decimal system. Please read there!
To end the input either RETURN or ESC has to be pressed. RETURN ends the enty of the number and the Carry flag is set to 1. But ESC does terminate and clears the Carry flag to 0.

Attention: This OS function jumps back only if the user either pressed RETURN or ESC. Please use this OS function only in screen MODE 2.

INPUT FUNCTION for 8 BIT VALUES (HEXADECIMAL) DISPLAYED on SCREEN

Short description: Input function for hexadezimal 8 bit values. ESC stops the input.

Label: BIT8_IN

ROM-number: A

Start address: &C31A

Jump in conditions: The cursor position, defined through RAM variable C_POS, will be used for displaying the entered value.

Jump out conditions: A = 8 bit value, entered by the user. Carry = 1.
If the Carry flag is cleared (NC) the OS function was ended through the ESC key. Then please ignore the entered value.

Manipulated: AF, BC, DE, HL, IY, AF', BC', DE', HL' and RAM variable C_POS, which contains the position of the software cursor.

Description: This OS function is used to enter 8 bit values. The actual cursor position is used. Values entered are interpreted in hexadecimal and NOT in decimal. They are shown as Mode 2 chars, so set screen mode 2 before calling it.

After the CALL of this OS function the user must enter two values (in the region between 0..9 and A..F). First the higher four bits, then the lower four bits (like you can assume). Shift and Control are ignored. You can hit DEL to delete the last entered 4 bit value. To end the input you just have to press the RETURN key. Then the OS function jumps back with the entered 8 bit value in the A register and with a set Carry flag.

In addition it is possible to finish this OS function by pressing the ESC key. In this case the Carry flag is CLEARED and the value in register A is not valid.

Well, ESC can't be used if you have already entered two keys, then you have to press DEL first, then ESC. Just try it.

Attention: This OS function jumps back if you have pressed a number key (0..9 and a..f) twice and then RETURN. Or it jumps back if you press ESC. Else the system waits for the next system crash or power down.

Switch the screen mode to 2 before using this OS function!

This OS function is used by the F_PORT high-level machine monitor.

INPUT FUNCTION for 8 BIT VALUES (DEZIMAL) DISPLAYED on THE SCREEN

Short description: Input function for dezimal 8 bit values. ESC stops the input.

Label: B8DIN

ROM-number: A

Start address: &FE41

Jump in conditions: The cursor position, defined through RAM variable C_POS, will be used for displaying the entered value.

Jump out conditions: A = 8 bit value, entered by the user. Carry = 1.
If the Carry flag is cleared (NC) the OS function was ended through the ESC key. Then please ignore the entered value.

Manipulated: AF, BC, DE, HL, IX, IY, BC', DE', HL', PIO, PSG and RAM variable C_POS, which contains the position of the software cursor.

Description: This OS function is used to enter 8 bit values. The actual cursor position is used. Values entered are interpreted in decimal. They are shown as Mode 2 chars, so set screen mode 2 before calling it.

After the CALL of this OS function the user must enter three digits (in the region between 0..9). First hundred, then tenner, then singles (like you can assume). Shift and Control are ignored. You can hit DEL to move the cursor to the left. To end the input you just have to press the RETURN key. Then the OS function jumps back with the entered 8 bit value in the A register and with a set Carry flag.

In addition it is possible to finish this OS function by pressing the ESC key. In this case the Carry flag is CLEARED and the value in register A is not valid.

Attention: This OS function jumps back if you press RETURN or ESC.
Else the system waits for the next or power outrage.
Switch the screen mode to 2 before using this OS function!
The maximum number you can enter is "255", higher numbers are not valid! In this case you have to enter again.

INPUT FUNCTION FOR 16 BIT VALUES DISPLAYED ON SCREEN

Short description: Input function for values between 0 and 65535.

Label: RB_16 or RBB_16

ROM-number: A

Start address: &FE3E (RB_16) or &FE38 (RBB_16)

Jump in conditions: The cursor position C_POS (RB_16) or one position before the cursor position C_POS-1 (RBB_16) will be used to enter the value.

Jump out conditions: A contains the value entered by the user and the Carry flag is set to 1. If the Carry flag is cleared (NC) the OS function was ended through the ESC key. Then please ignore the entered value.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY, PIO, PSG and the system RAM variable C_POS, which contains the cursor position.

Description: These two OS functions are used to enter 16 bit values (0-65535) in MODE 2. The actual cursor position (OS function RB_16) or the position one to the left (OS function RBB_16) will be used. In the latter case the character left to the cursor will be overwritten, but only when using the decimal system. This way an possible "&" character for the entry of hexadezimal numbers can be overwritten automatically.

Both OS functions RB_16 and RBB_16 jump either to B16IN or B16DIN, this depends on FutureOS using the hexadecimal or the decimal system. Please read there!

To end the input either RETURN or ESC has to be pressed. RETURN ends the enty of the number and the Carry flag is set to 1. But ESC does terminate and clears the Carry flag to 0.

Attention: This OS function jumps back only if the user either pressed RETURN or ESC. Please use this OS function only in screen MODE 2.

INPUT FUNCTION for 16 BIT values (HEXADEZIMAL) DISPLAYED on SCREEN

Short description: Input function for 16 bit values in the hexadecimal system. ESC stops the input.

Label: B16IN

ROM-number: A

Start address: &C38A

Jump in conditions: The cursor position (*=-> RAM variable C_POS) will be used for displaying the entered value.

Jump out conditions: DE' = 16 bit value, the user input.
If this OS function returns with a cleared Carry Flag (NC), then this is due to pressing ESC. In this case ignore the content of DE'.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL' and the RAM variable C_POS (contains cursor position).

Description: This OS function is an input function for 16 bit values. The actual cursor position (contained in C_POS) is used to display the entered values (format &12C9) on the screen (Mode 2). The hexadecimal system is used. Switch to screen mode 2 before calling this OS function.

After calling B16IN the user has to press four keys between 0..9 or a..f. These four keys compose a 16 bit value. The keys Shift and Control are ignored. You can use DEL to step backwards (try it). After entering a 16 Bit value you just have to finish the input through pressing RETURN. Then register DE' will contain the 16 bit input and the Carry flag will be set.

Furthermore you can press ESC at any time, and with pressing ESC you stop the input function. Then it will return with a cleared Carry flag and register DE' contains no valid value. If you have entered four keys and want to escape through pressing ESC, you have to press DEL first and then ESC. Just for your security.

Attention: This OS function returns only when you have entered a 16 bit value or by pressing ESC. There is no other exit.

The FutureOS machine monitor uses this OS function.

INPUT FUNCTION for 16 BIT VALUES (DEZIMAL) DISPLAYED on the SCREEN

Short description: Input function for 16 bit values in the decimal system. ESC stops the input.

Label: B16DIN

ROM-number: A

Start address: &FE44

Jump in conditions: The cursor position (*=-> RAM variable C_POS) will be used for displaying the entered value.

Jump out conditions: DE' = 16 bit value, the user input.
If this OS function returns with a cleared Carry Flag (NC), then this is due to pressing ESC. In this case ignore the content of DE'.

Manipulated: AF, BC, DE, HL, BC', DE', HL', IX, IY, PIO, PSG and the RAM variable C_POS (contains cursor position).

Description: This OS function is an input function for 16 bit values. The actual cursor position (contained in C_POS) is used to display the entered values (format "12345") on the screen (Mode 2). The decimal system is used. Switch to screen mode 2 before calling this OS function.

After calling B16IN the user has to enter five digits between 0 and 9. These five numbers compose a 16 bit value. The keys Shift and Control are ignored. You can use DEL to step backwards. After entering a 16 Bit value you just have to finish the input through pressing RETURN. Then register DE' will contain the 16 bit input and the Carry flag will be set.

Furthermore you can press ESC at any time, and with pressing ESC you stop the input function. Then it will return with a cleared Carry flag and register DE' contains no valid value.

Attention: This OS function returns only when you have entered a 16 bit value or by pressing ESC. There is no other exit.
The maximum number you can enter is "65535", higher numbers are not valid! Then you have to enter again.

CONVERSION OF ONE BYTE INTO TWO ASCII CHARS

Short description: One single byte will be converted into two ASCII characters. Later you can display these two chars on the screen, so you can show a hexadecimal byte.

Label: N_2_2C

ROM-number: A

Start address: &D58A

Jump in conditions: A = source-byte that should be converted to ASCII.
                          DE = pointer to the target address of the two generated ASCII chars.

Jump out conditions: A and (DE) contain the lower and (DE-1) the higher nibble of the source-byte. DE was increased by 1.

Manipulated: AF, BC and DE was increased by 1.

Description: If you want to show a byte (between &00 and &FF) on the screen you have firstly to convert it into two ASCII chars. Exactly that is what this OS function does. DE addresses an memory position, the high-nibble will be written into (DE). The following address (DE+1) will become the lower-nibble. The two generated ASCII chars can be printed on the screen immediately. Then you seen the source-byte on the screen in hexadecimal.

This OS function does NOT display the chars itself, it only converts them and saves them to memory.

Attention: DE must contain a valid value, else two bytes will be written somewhere in the RAM, this can cause trouble.

SHOW AND EDIT A STRING

Short description: With STED you can show, edit or generate a text string.

Label: STED

ROM-number: A

Start address: &FE7D

Jump in conditions:
B = upper border of usable characters. First char that is located below the highest usable ASCII char, &01-&FF.
C = lower border of usable characters, &00 to &FF.
DE = pointer to the first byte of the string which should be edited. Region: &0000 to &FFFF.
HL = pointer to the byte after the end of the string (&0000-&FFFF).

Jump out conditions: The string has been edited by the user.
                                A = &03 => Editing ended through key ESC!
                                A = &0E => Editing ended normally.

Manipulated: AF, BC, DE, HL, BC', DE', HL', IX, IY, PIO and PSG.

Description: With STED you can edit or create any MODE 2 text string. The string length only depends on the format of the screen. Before you call this OS function you load register DE with the address of the first byte of the string. And register HL must point to the memory address of the byte which follows the last byte of the string. That means that the length of the string is HL - DE.
Further you must tell STED which characters are allowed to be entered. Therefore register B has to be loaded with the number of the ASCII char which follows the highest allowed ASCII char. And register C must contain the ASCII value of the first allowed char.

Example: B = &80 and C = &20 =--> You can enter all ASCII characters in the region between &20 and &7F. (&20 and &7F inclusive).

While editing you can move the cursor with the arrow keys and the DEL key. To finish editing you just have to press RETURN or ENTER.
But you can stop editing through ESC too.
When the OS function jumps back register A contains a value which informs you which key has ended editing. If it is &0E, then RETURN was used, else if A contains &03 then ESC has terminated editing. The following lines could make it more clear:

Example:

```
 LD    BC,&8020            ;chars from &20 to &7F are legal
 LD    DE,&8400            ;Text string starts at address &8400
 LD    HL,&8420 + 1        ;and ends at address &8420 in RAM
 CALL STED                 ;edit string
 RRCA                      ;move bit 0 into the Carry flag
 JR    C,ESC               ;If bit 0 is 1, editing was terminated using ESC
```

Attention: The string is not allowed to be longer than the maximal displayable number of characters (that depends on the screen format).

If you have set your screen format (Mode 2 !!!) to 64 columns and 32 lines for example, the string is not allowed to be longer than nearly 2 KB. This subOS function only runs with screen mode 2. You have to set Mode 2. When calling this OS function register HL must be bigger than DE.

INVERT TWO MODE 2 CHARACTERS (MODE 1 CURSOR)

Short description: Two successive chars will be inverted. Mode 2 chars are used.

Label: CUR_INV

ROM-number: A

Start address: &D4EC

Jump in conditions: C_POS = cursor position of first char (2. follows)

Jump out conditions: Two Mode 2 chars have been inverted. Furthermore the first 64 KB RAM are switched in (standard RAM, config. &C0). Both ROMs (lower and upper) are switched on and screen Mode 2 was set.

Manipulated: AF, BC, HL, 16 bytes V-RAM, RAM configuration (to &C0) and the RAM/ROM-state (to &82).

Description: This OS function is used to show or (better said) to invert a cursor. This cursor comprises two Mode 2 characters or else one Mode 1 character. But since this OS function switches screen Mode 2 on, it should only be used under that mode.

You can call this OS function two times and the cursor will become what it has been before the first call of this OS function.

Despite this OS function is extremely fast, it will switch to standard RAM (&7FC0) and to screen Mode 2 with both ROMs switched in (&7F82).

Attention: The first 64 KB and both ROMs are switched on.

INVERT ONE MODE 2 CHARACTER (MODE 2 CURSOR)

Short description: One Mode 2 character (cursor) will be inverted.

Label: CUR_IV2

ROM-number: A

Start address: &D544

Jump in conditions: C_POS = cursor position of char to be inverted.

Jump out conditions: One Mode 2 char is inverted. The first 64 KB (RAM config. &7FC0) and Mode 2 are set. Both ROMs (&7F82) are switched on.

Manipulated: AF, BC, HL, 8 bytes V-RAM, RAM configuration (to &C0) and the RAM/ROM-state (to &82).

Description: This OS function allows to show a cursor under screen Mode 2. The screen position is selected through (C_POS). This screen position will be inverted through this OS function. It's like a software cursor for Mode 2.

You can call this OS function a second time and the screen position will become what it has been before the first call of this OS function.

Despite this OS function is extremely fast, it will switch to standard RAM (&7FC0) and to screen Mode 2 with both ROMs switched in (&7F82).

Attention: The first 64 KB and both ROMs are switched on.

SHOW HEADER OF A FUTURE-OS FILE

Short description: The header or icon of an FutureOS file-header of a file will be shown on the screen.

Label: SHED

ROM-number: A

Start address: &FE71

Jump in conditions: RAM variable C_POS contains the target position on the screen, where the data should be shown.
You have to set the screen to MODE 1 and 32 * 32 chars. Maybe you have to clear the screen.
A FutureOS file-header (128 byte) must be at address &B400 in RAM.

Jump out conditions: Head-line, Semi-graphic- or Full-graphic-Icon was printed on the screen at (C_POS)

Manipulated: AF, BC, HL, IX, IY and the video-RAM after C_POS.

Description: Well, under Amsdos every NON-ASCII file has a 128 byte file-header. This file-header is strongly expanded under FutureOS. It has additional information about the file and can contain a file-icon like known from other GUI-OS.

Before you can show a file-header you must load the RAM variable C_POS with the cursor position where the file-icon or headline should be shown. Further you must load a 128 Byte file-header to address &B400 in RAM - before calling SHED.

This OS function only displays the additional information of an FutureOS file-header (like icons) on the screen. Look jump in conditions!

SHED can show headlines, semi-graphic-icons (text-icons) or graphic icons (two formats, look icon description in big handbook).

Attention: Before you call SHED you must have loaded a FutureOS file-header at &B400. You must have set the screen mode to 1 and you must switch the screen format to 32 Mode 1 columns and 32 lines.

COPY REAL-TIME-CLOCK MANAGEMENT INTO THE RAM

Short description: Copy Real-Time-Clock-Manager (of the Dobbertin / dxs or M4 RTC) from ROM to the RAM.

Label: DOBIN

ROM-number: A

Start address: &FE6E

Jump in conditions: -

Jump out conditions: Some OS functions of the RTC-manager have been copied to the RAM at the addresses LUHR, RDUK and KOAS.

Manipulated: BC, DE, HL and the Flags.

Description: The real-time-clock (RTC) from Dobbertin / dxs is connected to the CPC like a ROM. To read data from the Dobbertin RTC the corresponding ROM needs to be switched on. Well, if the RTC is switched on, the FutureOS ROMs are off. Therefore you need functions to switch on the RTC's ROM number to be able to read the data and to switch the FutureOS ROMs back on again. Such OS functions must be located in the RAM.

The same is true for the RTC of the M4 board, which can be read using the M4 ROM.

The OS function DOBIN copies the RTC-manager from the ROM to the RAM. Only in the RAM they can work. There you can use LUHR to read data.

Normally DOBIN is called after every start-up of FutureOS. So normally the RTC-manager should be active in the RAM.

But if a program or applicaton has overwritten the RTC-manager (what really never should happen), the RTC-manager can be copied again in the RAM - this is done by DOBIN.

To read the date and time data from the Dobbertin / dxs or M4 RTC you can use LUHR. The OS function LUHR reads the data in the RAM at UHR_00 (see file #OS-VAR.ENG).

If the M4 expansion board is connected its RTC will be used. If there is an additional Dobbertin / dxs RTC it will be ignored.

Attention: This OS function only copies the manager for the Dobbertin RTC to the RAM. But no time or date is read (or written).

SHOW ASCII-CHAR EIGHT-FOLD ZOOMED

Short description: An ASCII-character is shown eight-fold zoomed in X and Y.

Label: PR2GR

ROM-number: A

Start address: &FE68

Jump in conditions: L = ASCII char from &00 to &FF. Mode 2 must be set active. The RAM-variables C_POS and MAX_CRX must contain valid values.

Jump out conditions: An eight-fold zoomed char is shown at (C_POS).

Manipulated: AF, BC, DE, HL, BC', HL', YL and the V-RAM.

Description: This OS function is used to show an 8-fold zoomed char on the Mode 2 screen. The char is printed at the actual cursor position C_POS which will not be altered. Right and under the cursor there should be eight columns and lines free. You can use every screen format (lines, columns), but the correct number of columns must be written in the RAM variable MAX_CRX.

SHOW A BIT-MATRIX - 8 BIT WIDTH

Short description: A bit-matrix, 8 bit in width, that can have any bit-height will be printed on the screen. The matrix is 8-fold zoomed.

Have a look at the similar OS function PR2GR.

Label: PR2GS

ROM-number: A

Start address: &FE65

Jump in conditions: HL = source address of matrix data. Mode 2 must be switched on. RAM-variables C_POS, MAX_CRX must contain valid number.
YL = Number of lines, height of the matrix. This is equal to the number of source bytes, beginning at HL.

Jump out conditions: Matrix shown on the screen beginning at (C_POS).

Manipulated: AF, BC, DE, HL, BC', HL', YL and the V-RAM.

Description: This OS function is very similar to PR2GR. But this OS function doesn't show a 8-fold zoomed character, it shows a 8-fold zoomed bit-matrix on the screen. The matrix can have any high, but the width is in any case 8 bit (every byte has 8 bit).

Before you call this OS function, you have to put the height of the matrix in lines (= number of bits) in register IY low (YL). You can use: DB &FD:LD L,10 to set YL to 10.

With that you can show the complete keyboard matrix. Further you have to load HL with the first byte of the matrix data. The matrix is shown at (C_POS), which is not altered.

Attention: The matrix is not allowed to leave the borders of the screen (check register IY low).

PRINT A 7 BIT CHARACTER ON THE PRINTER

Short description: A 7 bit value (&00-&7F / 0-127) will be sent to the printer. OS function runs with all CPCs.

Label: DRZ7

ROM-number: A

Start address: &D67B

Jump in conditions: A = character (&00-&7F/0-127) to send to printer.
PIO port B (I/O address &F5XX) must be set to input (printer status).

Jump out conditions: The zero flag gives information about the success of the operation:
Zero flag set (Z) => character was send correctly to the printer.
Zero flag reset (NZ) => printer was NOT ready!

Manipulated: F, BC and the printer port &EFXX.

Description: DRZ7 is a OS function which allows to send a 7 bit character to the printer. It works with all CPCs. Before calling this OS function you have to load register A with the value to be send to the printer. The eight Bit is ignored. DRZ7 returns the zero flag depending on the success of the OS function. If the value was send to the printer the zero flag is set. If the printer is NOT ready the zero flag is cleared.

Attention: PIO port B must be switched to INPUT, else this OS function can't sense the status of the printer. Normally port B is set correct.

WAIT UNTIL THE PRINTER IS READY

Short description: This OS function waits until the printer is ready (to get the first / next character).

Label: XW_DR

ROM-number: A

Start address: &D70D

Jump in conditions: -

Jump out conditions: The printer is ready to receive a character.

Manipulated: AF, BC and PIO status (Port B set to INPUT).

Description: You can only send a character to the printer if the printer is ready to receive characters. This OS function XW_DR waits until the printer status becomes 'READY' then it returns. The printer is ready if bit 6 of PIO port B is cleared.

XW_DR will not return until the printer is not ready. Afterwards you can send one character to the pritner.

Attention: Beware! If the printer is for example switched off XW_DR will NOT RETURN. Because it waits for 'PRINTER READY'

PRINT ONE 8 BIT CHARACTER (6128PLUS ONLY)

Short description: One 8 bit value will be sent to the printer. Plus!

Label: DRZP8

ROM-number: A

Start address: &D657

Jump in conditions: A = value (&00 - &FF / 0 - 255) to be printed.
PIO port B (address &F5XX) must be set to input (status of printer).

Jump out conditions: The Zero Flag reports about the success:
Zero Flag set   (Z)   => Character was sent to the printer. --> Good!
Zero Flag reset (NZ)   => Printer was / is NOT ready! -------> Just wait!

Manipulated: F, BC and the printer port &EFXX.

Description: DRZP8 allows to send one 8 - bit value / character to the printer. But it only works with the 6128 Plus!!! Before you call DRZP8 you have to load register A with the value that should be printed.

If all works pretty and the printer has got the value, then DRZP8 will return with the Zero flag set to 1. If DRZP8 returns with a cleared Z flag then the printer was (is) not ready to read a character.

Attention: Before you call DRZP8 PIO port B must be set to input.

PRINT ONE 8 BIT CHARACTER (CPC OLD GENERATION ONLY)

Short description: One 8 bit value will be sent to the printer. CPCoG!

Label: DRZO8

ROM-number: A

Start address: &D66D

Jump in conditions: A = value (&00 - &FF / 0 - 255) to be printed.
PIO port B (address &F5XX) must be set to input (status of printer).

Jump out conditions: The Zero Flag reports about the actions success:
Zero Flag set   (Z)   => Character was sent to the printer. --> Good!
Zero Flag reset (NZ)   => Printer was / is NOT ready! -------> Just wait!

Manipulated: F, BC and the printer port &EFXX.

Description: DRZO8 is a OS function which sends one 8 bit character to the printer port of the CPCs old generation (464/664/6128) with installed 8-bit Printer-Patch (connection between PIO and Bit 8, see extra doc).

Register A must be load with the char that should be print. Then call DRZO8. When DRZO8 returns the Z flag is set or cleared depending on the result of the action. If the char was printed properly Z is set, and if the printer is NOT ready the Z flag is cleared.

Attention: PIO port B must be set to input (this is standard).
Beware! This OS function uses the 8 bit PIO <--> printer hardware patch!

SEND A STRING TO THE PRINTER (7/8 BIT - 6128 PLUS / all CPCs)

Short description: Three OS functions for sending strings to the printer.

Labels:
o DRS7 : Print a 7 bit string, all CPCs.
o DRSP8 : Print a 8 bit string, PLUS CPCs.
o DRSO8 : Print a 8 bit string, CPCs 464/664/6128 (oG).

ROM-number: A

Start address: &D6F0 (DRS7) // &D692 (DRSP8) // &D6C5 (DRSO8)

Jump in conditions: HL = start address of the string (&00 marks end).

Jump out conditions: The OS function(s) jumps back after sending the whole string to the printer.

Manipulated: AF, BC, DE, HL and the printer port.

Description: These three OS functions can be used to send a string to the printer. Such a string must be terminated through a &00 byte.

Depending on the type of CPC (Plus or old generation) and printer port (7 or 8 data bits) you have to choose the right OS function. A program can read the configuration bytes to decide which type of CPC and printer interface is used.

Before calling such a OS function you have to load register HL with the start address of the string (that is to print). The end of the string is marked through the byte &00. These OS functions don't return until the complete string is send to the printer.

It is important to sense the status of the printer before calling one of the above OS functions. Example:

```
TEST_PRINTER LD  BC,&F782     ;PIO control register
             OUT (C),C        ;set port B to input

 LD  B,&F5      ;PIO port B
 IN  A,(C)      ;read printer status
 BIT 6,A        ;test printer status

 JR NZ,PRINTER_NOT_READY

 ;or ...

 JR Z,PRINTER_READY
```

Attention: The printer status MUST be 'READY' before calling one of the above OS functions, else they may never return.

HD20 - COMPUTE NUMBER OF FREE ENTRIES OF ONE PARTITION

Short description: This OS function computes the number of free DIRectory entries of one 5 MB partition of the Dobbertin HD20 hard disc.

Label: EFED_HD

ROM-number: A

Start address: &D71B

Jump in conditions: REG08_1 = Partition 8...11 (equals icon I, J, K, L).
The 32er DIR(ectory) of that partition must have been read before.

Jump out conditions: BC = &0020 // HL = &8000
DE = Number of free DIR entries of the partition (REG08_1).
The 32er DIR of the partition is banked in beginning at &4000.

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

Description: This OS function calculates the number of free DIR entries of one partition of the Dobbertin hard-disc HD20.

You have to put the number of the partition (8, 9, 10 or 11) to the RAM variable REG08_1 before calling EFED_HD. Partitions I, J, K and L are symbolized through numbers 8, 9, 10 and 11 ----> NOT through 0 – 3 (like known from the HD20 loading OS functions).

A further prerequisite is that the 32er DIR (physical 16 KB directory) of the corresponding HD20 partition has been read and buffered in RAM before EFED_HD can be used.

This OS function returns the number of free DIR entries in register DE. That number can lie between &0000 and &0200, because the 16 KB DIR of one partition contains 512 entries.

Attention: Before calling this OS function, the Dobbertin hard-disc HD20 must be active and its DIR must have been read and buffered in RAM.

HD20 - GENERATE BLOCK-OCCUPANCY-TABLE OF ONE PARTITION

Short description: A block-occupancy-table for one hard-disc partition will be generated.

Label: BBTG_HD

ROM-number: A

Start address: &D73F

Jump in conditions: REG08_1 = Partition 8..11 (equals I, J, K or L).
The 32er DIR(ectory) of that partition must have been read before.

Jump out conditions: Block-occupancy-table is located in RAM at &B000.
The 32er DIR of the partition is banked in beginning at &4000.

Manipulated: AF, BC, DE, HL, HL', IX, the RAM - status and RAM between &B000 and &B7FF.

Description: This OS function computes the BOT (Block-Occupancy-Table) of one partition of the Dobbertin HD20 hard-disc. The BOT will be written in RAM beginning at &B000. Normally (5 MB partition) it ends at &B50D. In every case it ends below &B7FF.

The first four 4 KB blocks (0 - 3) contain the 16 KB DIR of the HD20 partition. So, if you search for free blocks, you should start your search with the fifth block, block &0004 = 4. In such an BOT the block byte for Block &0000 is located at address &B000, the byte for block &0001 is at &B001 and so on. The last block-byte (block &050D) is therefore located at address &B50D.

An occupied block is symbolized through Block-Byte &FF, a free block is marked through Block-Byte &00.

Before you call BBTG_HD you have to write the number of the partition to the RAM in RAM variable REG08_1. The four partitions I, J, K and L are symbolized through the numbers 8, 9, 10 and 11 (NOT 0-3).

When this OS function returns the BOT has been written to RAM beginning at &B000. After the end of the table &00 Bytes will follow up to &B7FF. The 32er DIR (16 KB physical directory) of the HD-20 partition remains banked in between &4000 and &7FFF.

Attention: Before calling this OS function, the Dobbertin hard-disc HD20 must be active and its DIR must have been read and buffered in RAM.

HD20 - GENERATE DIRECTORY-ENTRY OF A FILE

Short description: Generates the directory-entry of a file which shall be saved to the HD20 hard-disc.

Label: EGEN_HD

ROM-number: A

Start address: &D7A7

Jump in conditions: 32er DIR must be banked it at &4000.
BC = number of blocks of the file whose entry should be generated.
REG16_8 = User(1), Name(8), Extent(3) of the file. 12 Bytes.
Beginning at &B000 there must be a BOT of the partition
(see BBTG_HD)

Jump out conditions: The file has got its entry in the DIRectory.
Now, the physical 16 KB 32er DIR of the partition is unsorted.

Manipulated: AF, BC, E, BC', DE', HL', XL and the DIR is not sorted.

Description: If a file should be saved to hard-disc, then that file must become previously one or more DIRectory-entries (depending on the size of the file). Only AFTER THAT you are able to generate the file-save-table (using HD_TAB) which is needed to write the data of a file to the hard-disc (using SHD4 or SHD6).

This OS function EGEN_HD is used to generate the DIR entry/entries of a file which should be saved after that. Before calling this OS function the following must be accomplished:

o The DIR (in which the entry should be written) must be banked in between &4000 and &7FFF. This is made by BBTG_HD for example!

o The BOT (Block-Occupancy-Table) of the partition (generated through BBTG_HD) must be located at &B000 in RAM.

o The register BC must contain the number of blocks of the file. That means the block-number of the file that should be saved later.

o Beginning at RAM variable REG16_8 there must be 12 bytes in RAM - they contain User (1 byte), Name (8 bytes) and Extension (3 bytes).

When the OS function was called and returns, there will be an DIR entry for the specified file (name at REG16_8) in the DIR.

One DIR entry is big enough to cover 32 KB. For each following 32 KB one further DIR entry will be used. The Dobbertin HD20 hard disc has blocks of 4 KB and every block number hat 16 bit.

After calling this OS function the DIR is not any longer sorted. If you want to sort the DIR (what would be an good idea) you can follow these steps: switch ROM C on, then call the OS functions SSB0, ISWS and RRB0 at last.

While generating DIR entries the number of records will be rounded up to the next 4 KB border. For example: A 17 KB file will use 20 KB of the hard-disc space (look 4 KB blocking).

Attention: When the OS function returns the DIR is unsorted. Therefore use / call SSB0, ISWS, RRB0 (all in ROM C) in that sequence.

SAVE A FILE (PARTIAL) FROM THE SHORT-TIME-MEMORY TO THE HD20

Short description: A file will be saved (partial) to the hard-disc HD20. The source of the file is expansion RAM: STM (short-time-memory)

Label: TSDHD (first call) and TSEHD (rest of file)

ROM-number: A

Start address: &D88D (TSDHD), &D87E (TSEHD)

Jump in conditions:

TSDHD:
A = HD20 partition (8..11) where the file should be saved.
REG_BC1 = file length in KB.
REG16_8 = User(1), Name(8) and Extent(3) of the file (12 bytes).

TSEHD:
It is not permitted to change the RAM variables REG_AF1, REG_DE1,
REG_HL1, REG_IX and REG16_8 to REG16_8+12 since the last call of TSDHD (or TSEHD).

Jump out conditions: The success of this operation is provided in the A register:
A = &FF => The complete file was saved to the hard-disc. Good!!!
A = &F0 => The file was saved PARTIAL to the hard-disc, the rest of the file must be saved using OS function TSEHD.

Errors:
A = &00 => No DIRectory was read previously from the HD20 partition.
A = &01 => The chosen partition is not tagged.
A = &02 => The file-length is 0 KB. break-off!
A = &04 => The target-DIR has not enough free DIR-entries.
A = &05 => No STM is used in E-RAM, there is nothing to save.

Manipulated, using TSDHD and TSEHD: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY, RAM variables REG08_0,_1, REG_DE1(low) and REG_PC(low).

Manipulated, only using TSDHD: RAM variables TURBO_A to M, REG16_0 to 9, REG32_1,2, REG_AF1, REG_HL1, REG_IX, the RAM from &B000 to &BFFF.
Further the RAM configuration and the DIRectories have been changed.

Description: This OS functions are the hard-disc counterparts to the OS functions TEILA and TEILB. While TEILA loads a file (partial), the OS functions TSDHD / TSEHD are used to save a (previously loaded) file (partial). This two OS functions are only used to save to the Dobbertin hard-disc HD20. To save a file (partial) to floppy disc, just use the OS functions TEISI / TEISK.

The file will be saved completely or in parts. First you call TSDHD. TSDHD generates the DIRectory entry and saves all short-time-memory expansion RAM blocks to hard-disc. After the OS function returns, an error / success code is given in register A.

If TSDHD returns with the byte &FF in register A the file was saved completely. If A contains the value &F0 all expansions RAMs (marked as short-time-memory) have been saved, but the file is longer than the free RAM. In this case you have to preserve some RAM variables (look above). Now you first have to load the rest of the file (using TEILB), then you can save this rest of the file using TSEHD to hard-disc. May you have to do this some times. This depends on the file length and the amount of expansion RAM.

Well, these OS functions update the DIR (new entries added), but they don't save it to the hard-disc. This makes sense if you want to save some more files to the hard-disc. The hard-disc DIR is 16 KB long!

You can save the hard-disc DIR by using OS function SIDIR.

Attention: You have to save the updated DIR using SIDIR. Before you call SIDIR it could make sense to mark one further 16 KB E(xpansion)-RAM (directly before the actual DIR buffer RAM) as a DIR buffer. The generation of new DIR entries increase the amount of needed E-RAM.

BEWARE: TSDHD and TSEHD are only usable with the Dobbertin hard-disc HD20. OS functions to save files (partial) to disc are located in ROM C.

CHECK IF THE SYMBiFACE II IS CONNECTED

Short description: This function checks if the SYMBiFACE II expansion card is connected to the CPC or not (No check for CPC-IDE or X-MASS).

Label: T_SF

ROM-number: A

Start address: &FE4D

Jump in conditions: -

Jump out conditions: Z-Flag is set -----> SYMBiFACE II is connected
Z-Flag cleared --> SYMBiFACE II isn't connected

Manipulated: AF, BC and the Millenium byte of the SF2-RTC is set to 20

Description: This function can be used to check if the SYMBiFACE II expansion card is currently connected to the CPC. Depending on the existence of the SF2 the Z flag will be set. If an SF2 is connected and active then the Zero flag will be set to one (Z), else it will be cleared (NZ).

This function seperates between a CPC-IDE or X-MASS on the one hand and the SYMBiFACE II on the other. Therefore it checks the availability of the Real Time Clock (RTC). The CPC-IDE or X-MASS will not be recognized and the Zero flag will be cleared.

Attention: The CPC-IDE or X-MASS expansions will not be recognized.
This function sets the Millenium-Byte of the RTC to 20.

CHECK IF THE SYMBiFACE III IS CONNECTED

Short description: This function checks if the SYMBiFACE III expansion card is connected to the CPC or not.

Label: T_SF3

ROM-number: A

Start address: &D875

Jump in conditions: -

Jump out conditions: Z-Flag is set ---> SYMBiFACE III is connected
Z-Flag cleared --> SYMBiFACE III isn't connected

Manipulated: AF and BC

Description: This function can be used to check if the SYMBiFACE III expansion card is currently connected to the CPC. Depending on the existence of the SF3 the Z flag will be set. If an SF3 is connected and active then the Zero flag will be set to one (Z), else if will be cleared (NZ).

The echo function of the SF3 is used to determine its existence.

Attention: -

WRITE TIME AND DATE TO THE RTC OF THE SYMBiFACE II

Short description: This function write a new time and date to the Real Time Clock (RTC) of the SYMBiFACE II.

Label: S_SFRT

ROM-number: A

Start address: &FE50

Jump in conditions: HL = 7 Bytes data (time, date; in BCD format) Sequence of the 7 data bytes: second, minute, hour and day-of-week, day, month, year.

Jump out conditions: Time data has been written to the RTC of the SF.

Manipulated: AF, BC, DE, HL and the RTC of the SYMBiFACE II.

Description: S_SFRT is used to write time and date into the Real-Time-Clock of the SYMBiFACE II. Register HL point to a 7 byte data block, which contains the following data:
- Second   &00 - &59
- Minute   &00 - &59
- Hour   &00 - &23
- Day of Week &01 - &07 (1 = Monday)
- Day   &00 - &31
- Month   &01 - &12
- Year   &00 - &99

The format of that seven bytes is BCD. These 7 data bytes can be located anywhere in the memory.

FutureOS uses these 7 bytes in that array at UHR_SEK in the system variables. If you want to set the RTC manually, you can use that array of bytes for that purpose too.

Attention: Only the RTC of the SYMBiFACE II will be changed. No other connected RTC will be altered.

WRITE TIME AND DATE TO THE RTC OF THE SYMBiFACE III

Short description: This OS function writes new time and date to the RTC of the SYMBiFACE III.

Label: S_SF3RTC

ROM-number: A

Start address: &FE2C

Jump in conditions: There need to be valid time and date bytes in the system RAM starting at the system variable UHR_SEK. The BCD format is used.

Jump out conditions: Time and date have been written into the SF3 RTC.

Manipulated: AF, BC, DE, HL and the RTC of the SYMBiFACE III.

Description: The OS function S_SF3RTC is used to write time and date into the RTC of the SYMBiFACE III.
Time and date will be read from the system variable RAM beginning at UHR_SEK. The date block looks like this:
- Second   &00 - &59
- Minute   &00 - &59
- Hour   &00 - &23
- Day of Week &01 - &07 - not used
- Day   &00 - &31
- Month   &01 - &12
- Year   &00 - &99

The data format of this seven bytes is BCD.

Attention: Only the RTC of the SYMBiFACE III will be changed. No other connected RTC will be altered.

READ TIME AND DATE OF THE SYMBiFACE II RTC INTO RAM

Short description: The data (f.e. time and date) of the SYMBiFACE II's Real-Time-Clock (RTC) get read into the RAM.

Label: R_SFRT

ROM-number: A

Start address: &FE53

Jump in conditions: HL = Pointer to a buffer of 10 bytes of RAM

Jump out conditions: The buffer (defined by HL before) is filled with the data of the RTC. Their sequence is: Year, Month, Day, Day of week, Alarm-Hour, actual Hour, Alarm-minute, actual minute, Alarm-second, actual second.
Carry cleared --> Data have been read properly!
Carry is set -----> An error occurred, f.e. internal update of the RTC!

Manipulated: AF, BC, D, HL and 10 bytes beginning at former HL in RAM.

Description: R_SFRT reads the data of the Real-Time-Clock of the SYMBiFACE II into the RAM. The target address is defined by register HL (before calling this function). After the return of this function the 10 bytes of data from the RTC will be located in RAM at the buffer previously defined through register HL. The sequence of the data bytes is the following:
- Year   &00 - &99
- Month   &01 - &12
- Day   &01 - &31
- Day of the week   &01 - &07 (1 = Monday)
* Hour Alarm   &00 - &23
- Actual Hour   &00 - &23
* Minute Alarm   &00 - &59
- Actual Minute   &00 - &59
* Second Alarm   &00 - &59
- Actual Second   &00 - &59

The data is provided in BCD format again (see OS function before). The success of this function can be assessed by checking the Carry flag. If the function returns and the Carry flag is cleared, then all data have been read correctly. But if the Carry flag is set after the return of the function, then there was an error. In this case it's most likely that the RTC just did an 'internal Update'. Now you can call this OS function just a second time and most likely everything will be alright.

Attention: After calling this function please check the Carry flag for potential errors.

READ TIME AND DATE OF THE SYMBiFACE III RTC INTO RAM

Short description: The data (time and date) of the SYMBiFACE III's RTC will be read into the system RAM (at UHR_SEK).

Label: R_SF3RTC

ROM-number: A

Start address: &FE2F

Jump in conditions: -

Jump out conditions: The seven system variables beginning at UHR_SEK have been filled with time and date from the SF3 RTC.

Manipulated: AF, BC, DE, HL and seven bytes beginning at UHR_SEK.

Description: The OS function R_SF3RTC reads the data of the RTC of the SYMBiFACE III into the system RAM. Its written beginning at the system variable UHR_SEK and the subsequent bytes.
After the return of this function the seven bytes of data from the RTC will be located in system RAM at UHR_SEK:
- Actual Second &00 - &59
- Actual Minute &00 - &59
- Actual Hour &00 - &23
- Day of the week &01 - &07 - unused!
- Day &01 - &31
- Month &01 - &12
- Year &00 - &99

The data is provided in BCD format again (see previous OS function).

Attention: -

TRANSFER DATA of the SYMBiFACE II-RTC INTO FutureOS RAM VARIABLES

Short description: Data previously read from the SF 2-RTC using R_SFRT will be converted into FutureOS data format and written to RAM beginning at RAM variable UHR_SEK.

Label: K_SF2O

ROM-number: A

Start address: &FE56

Jump in conditions: HL points to 10 data bytes of the SF 2 RTC. Their sequence is: Year, Month, Day, Day-of-Week, Alarm-hour, actual hour, Alarm-minute, actual minute, Alarm-second and actual second.

Jump out conditions: 7 time / date bytes written to RAM at UHR_SEK

Manipulated: F, BC, DE and HL

Description: If the OS function R_SFRT is used to read 10 data bytes from the Real-Time-Clock (RTC) of the SYMBiFACE II to RAM, then these bytes have a different sequence than the time/date bytes of FutureOS.

This OS function K_SF2O converts the previously read time/date bytes of the SF 2-RTC into FutureOS time/date format.

Before calling this OS function the register HL must be loaded with the address of the 10 bytes time/date data previously read from the RTC.

After the return of this function there will be 7 bytes (Time, Day-of-Week, Date) written to RAM, beginning at FutureOS system RAM variable UHR_SEK.

Attention: Hour, minute and second of the Alarm time of the RTC will not be converted.

SET THE REAL-TIME-CLOCK OF THE SYMBiFACE II TO BCD-FORMAT

Short description: The internal time format of the Real-Time-Clock (RTC) of the SYMBiFACE II will be set to BCD encoding.

Label: RTC2BC

ROM-number: A

Start address: &FE59

Jump in conditions: -

Jump out conditions: Zero-Flag set => BCD was already active Zero-Flag cleared => RTC now uses BCD format

Manipulated: AF, BC, DE, HL and the SF 2-RTC

Description: The Real-Time-Clock (RTC) of the SYMBiFACE II will be set to BCD encoding. The RTC can handle BCD and binary encoding. FutureOS uses BCD encoding, since it's more easy and efficient to handle than binary encoding. Furthermore the RTCs of others (f.e. Dobbertin) can only use BCD encoding. Therefore FutureOS manages all its time & date variables in BCD format. The time & date variables are located in the system RAM variables beginning at UHR_00.

Attention: The switch to BCD format of the SYMBiFACE II RTC may cause problems in non-FutureOS programs which are not capable of using BCD encoding or of switching back to binary format.

WRITE ALERT TIME TO THE RTC OF THE SYMBiFACE II

Short description: The alert time (hour, minute, second) of the RTC of the SYMBiFACE II will be set.

Label: S_SFWZ

ROM-number: A

Start address: &FE4A

Jump in conditions: HL = Pointer to three bytes alarm data, encoded in BCD format.
Sequence: Second (&00-&59), Minute (&00-&59), Hour (&00-&23).

Jump out conditions: The Alert time of the SYMBiFACE II-RTC is set.

Manipulated: AF, BC, DE, HL and SF II-RTC (RTC is set to BCD format!)

Description: The internal alert time of the Real Time Clock (RTC) of the SYMBiFACE II will be programmed.
When calling this OS function the register HL must point to an address in RAM at which a three byte data block starts. This data block has the following structure:
- Second   &00-&59
- Minute   &00-&59
- Hour   &00-&23

This three bytes are encoded in BCD format as usual.

Attention: The programming of the Alert Time of the SYMBiFACE II will not lead automatically to the generation of an alarm at corresponding time under FutureOS.
Therefore you have to set the desired time using the Alert Time Icon of the Turbo Desktop.

GET DATA FROM THE SYMBiFACE II PS/2 MOUSE

Short description: This OS function requests position data from the PS/2 mouse of the SYMBiFACE II expansion card.

Label: R_PS2

ROM-number: A

Start address: &FE32

Jump in conditions: -

Jump out conditions: Data for X-, Y-coordinates, fire buttons and mouse-wheel have been read. If there is no SF2 PS2 mouse connected of if it hasn't been used then registers DE and IY contain the value &0000.
If the PS2 mouse is connected and has been moved then data is accessible in the registers D, E, IY (low) and IY (high):
D = fire buttons %1,1,0, Back,Fwd, F2,F1,F0 (1=ON)
E = mouse wheel &01-&1F (positive) or &FF-&F0 (negative)

YL = X-coordinate
 = %000 00001 (+1) to %000 11111 (+31) - Mouse moved to the right
 = %111 11111 (-1) to %111 00000 (-31) - Mouse moved to the left

YH = Y-coordinate
 = %000 00001 (+1) to %001 00000 (+32) - Mouse moved down
 = %111 11111 (-1) to %111 00000 (-31) - Mouse moved up

Manipulated: AF, BC, DE, H and IY

Description: The OS function R_PS2 can be used to request data from the PS/2 mouse of the SYMBiFACE II. If there's no PS/2 mouse connected then both double registers DE and IY will reteuren the value &0000.
This is also the case if there is no SF2 connected at all.
If data was successfully retrieved from the SF2 PS2 mouse then register D will contain the five possible fire buttons in bits 4-0.
Register E will contain the data from the mouse wheel. And registers IY (low) and IY (high) contain the data for the X- and Y-coordinates respectively.
Negative values are provided as twos complement. For example -1 will be given as 11111111 (binary). This way the provided value can be directly added to the actual coordinate.

Attention: Using T_SF can be used to check if the SF2 is connected to the CPC and with it probably the PS2 mouse.
The PS2 mouse must be connected to the SF2 before the SF2 (and subsequently the CPC) are turned on. Else it can't be used because the SF2 doesn't support 'hot plug & play'.

GET DATA FROM THE SYMBiFACE III USB MOUSE

Short description: This OS function requests position data from the USB mouse of the SYMBiFACE III expansion card.

Label: R_USB3

ROM-number: A

Start address: &FE29

Jump in conditions: -

Jump out conditions: Data for X-, Y-coordinates, fire buttons and mouse-wheel have been read. If there is no SF3 USB mouse connected of if it hasn't been used then registers DE and IY contain the value &0000.
If the USB mouse is connected and has been moved then data is accessible in the registers D, E, IY (low) and IY (high):
D = fire buttons %?, ?, ?, Back, Fwd, F2, F1, F0 (1=ON)
E = mouse wheel 1-100 (positive) or -1 to -100 (negative)

YL = X-coordinate
 = %000 00001 (+1) to %011 11111 (+100) - Mouse moved to the right
 = %111 11111 (-1) to %100 00000 (-100) - Mouse moved to the left

YH = Y-coordinate
 = %000 00001 (+1) to %010 00000 (+100) - Mouse moved down
 = %111 11111 (-1) to %110 00000 (-100) - Mouse moved up

Manipulated: AF, BC, DE and IY

Description: The OS function R_USB3 can be used to request data from the USB mouse of the SYMBiFACE III. If there's no USB mouse connected then both double registers DE and IY will returen the value &0000.
This is also the case if there is no SF3 connected at all.
If data was successfully retrieved from the SF3 USB mouse then register D will contain the five possible fire buttons in bits 4-0.
Register E will contain the data from the mouse wheel. And registers IY (low) and IY (high) contain the data for the X- and Y-coordinates respectively.
Negative values are provided as twos complement. For example -1 will be given as 11111111 (binary). This way the provided value can be directly added to the actual coordinate.

Attention: Using T_SF3 can be used to check if the SF3 is connected to the CPC and with it probably its USB mouse.
The USB mouse should be connected to the SF3 before the SF3 (and subsequently the CPC) are turned on.

DECOMPRESSION OF A ADVANCED OCP ART STUDIO *.SCR FILE

Short description: These two OS functions are both used to decompress previously compressed screen files of the Advanced OCP Art Studio.

Label: OCPC0 and OCPXX

ROM-number: A

Start address: &C18C (OCPC0) and &C18F (OCPXX)

Jump in conditions: HL = Source address of the compressed OCP picture, which has been previously loaded into RAM.
When using OCPXX the register DE must contain in addition the target address at which the decompressed picture (16 KB) shall be stored.
When using OCPC0 the decompressed picture will be stored at &C000.

Jump out conditions: Zero-Flag set --> The 16 KB long decompressed picture has been stored beginning at address &C000 (using OCPC0) or at the address previously defined by register DE (using OCPXX).
Zero-Flag cleared --> There was an error in the decompression!

Manipulated: AF, BC, DE, HL, IX, IYL and RAM (decompressed picture)

Description: This two OS functions are used for the decompression of a picture file (*.SCR) which was (saved and) compressed by the program Advanced OCP Art Studio.

First you have to load the compressed screen file into the RAM. Then load the register HL with its start address (start of the picture in RAM).

When using the OS function OCPXX the register DE must be loaded with a valid target address in RAM. When using OCPC0 the target address will always be &C000. After providing the source address (HL), and if needed the target address (DE) you can call the OS function OCPC0 or OCPXX.

When you have been using OCPC0 the uncompressed picture will be found in RAM beginning at address &C000 (and ending at &FFFF).

If the picture shall be decompressed to another address you will use the OS function OCPXX. In this case you need to load DE with the target address. After the return of OCPXX there will be 16 KB grapic data beginning at the address prviously defined by DE.

The Zero-Flag will inform you if the decompression was successful (the Zero-Flag is set) or if there was an error (Zero-Flag cleared).

Attention: When using OCPXX instead of OCPC0 the register DE must be loaded with a target address. Beginning at this address there must be 16 KB space for the graphic data.

QUICK 8 BY 8 BIT DIVISION

Short description: This OS function divides an 8 bit integer value by another 8 bit integer value.

Label: DIV88

ROM-number: A

Start address: &C1DE

Jump in conditions: D = Dividend (the number which gets divided)
                          E = Divisor (the number by which "D" gets divided)

Jump out conditions: D = Quotient / Ratio (D/E)
                          A = Rest / Carryover (which can't be divided)

Manipulated: AF, D

Description: This OS function allows to divide an integer 8 bit value by another 8 bit integer value in a quick way.
Before calling DIV88 the Dividend must be loaded in register D and the Divisor must be provided in register E.
The OS function divides the content of register D by the content of register E.
When returning DIV88 provides the Quotient (Ratio) of the division D/E in the register D. If there is a carryover (ramainder) it will be provided in register A (accumulator).

Attention: This is an integer division. And for the CPC it is probably the fastest way to divide 8 bit integer values.

EDIT MEMORY IN HEXADECIMAL-MODE

Short description: A block of &01E0 bytes memory will be edited in the hexadecimal mode. This block must already be displayed on screen.

Label: EDIT

ROM-number: A

Start address: &FE5C

Jump in conditions:
- Register IY points to the start address to be edited.
- The RAM variable MON_ROM contains either &82 or &86. It selects if the lower ROM or RAM shall be banked in (&0000-&3FFF) while editing.
- The RAM variable MON_RAM contains the RAM select, that's either &C0 or &C4, &C5, &C6 and so on, depending if the first 64 KB shall be used or expansion RAM (&4000-&7FFF) while editing memory.
- The memory block to be edited must already have been displayed on screen (using OS function F_DUMP).

Jump out conditions: The memory block has been edited by the user.

Manipulated: AF, BC, HL, AF', IX, PIO and PSG

Description: This function can be used in combination with OS function F_DUMP to edit a block of memory. It can be ROM, RAM or expansion RAM.

When using a CPC 6128 Plus the ASIC RAM can be edited too.

Attention: -- more information will be provided soon --

 - Please see next page for an CODE example -

Example of Editing Memory using F_DUMP and EDIT

```
  CALL S68X30                    ;Set Screen to 68 characters and 30 lines!

  LD HL,&4000:LD (REG16_1),HL  ;Start address to be edited


ED_L0 LD HL,(REG16_1):CALL F_DUMP ;DUMP ROM (in E-RAM) - 480 Bytes

  CALL XWART                     ;Wait until keyboard is free

ED_L1 LD HL,&0804:CALL HOLE1TS:JP Z,ED_EX ;ESC!

  CALL H_JC
  RRCA:JR NC,ED_UP ;Up
  RRCA:JR NC,ED_DN ;Down
  RRCA:JR NC,ED_UP ;Left
  RRCA:JR NC,ED_DN ;Right
  RRCA:JR NC,ED_ED ;Fire 0 (Edit)
  RRCA:JR NC,ED_EX ;Fire 1 (Exit)
  JR ED_L1


ED_UP LD HL,(REG16_1):LD DE,&01E0:SBC HL,DE ;Up/Left

  LD A,H:CP A,&40:JR NC,EDUPO ;HL >= &4000
  LD HL,&7E20

EDUPO LD (REG16_1),HL:JR ED_L0


ED_DN LD HL,(REG16_1):LD DE,&01E0:ADD HL,DE ;Down/Right

  LD A,H:CP A,&80:JR C,EDUPO ;HL < &7FFF
  LD HL,&4000:JR EDUPO


ED_ED ;Fire 0

  LD IY,(REG16_1):CALL EDIT ;EDIT ROM (in E-RAM) - 480 Bytes
  JR ED_L0


ED_EX RET
```

SWITCH OFF ALL SOUND CHANNELS OF THE PSG

Short description: All kind of sound output of the PSG will be switched off immediately.

Label: MAUS

ROM-number: A

Start address: &FE5F

Jump in conditions: -

Jump out conditions: Sound output is switched off.

Manipulated: AF, BC, DE (= &3F07) and PIO, PSG

Description: The OS function MAUS (Musik AUS) is used to switch off all kinds of sound output generated by the PSG. All three channels get switched off and their loudness is set to zero.

Attention: It will be quiet on earth.

SEND A DATA BYTE TO A REGISTER OF THE PSG

Short description: A data byte will be written into a PSG register.

Label: S_PSG

ROM-number: A

Start address: &FE62

Jump in conditions: D = Data byte for the PSG
                          E = Number of the PSG register (target)

Jump out conditions: The byte from register D has been written to the PSG register defined by register E.

Manipulated: AF, BC, PIO and PSG

Description: The OS function S_PSG allows to write any data byte to any register of the PSG. All registers of the PSG from &00 up to &0F can be used.

Attention: After this OS function has finished the status register of the PIO has been changed. Furthermore PIO port C has been set to &00.
Thereforem the tape recoreder has stopped if it was running before and the line of the keyboard matrix has been set to zero.

FIND A FREE BLOCK OF 16 KB IN THE EXPANSION RAM

Short description: One free 16 KB block of RAM will be searched within the first 512 KB of expansion RAM. It will be booked as "used" in the corresponding system variables of the OS.

Label: EFER

ROM-number: A

Start address: &D9C1

Jump in conditions: -

Jump out conditions: The register A will contain the physical RAM select for the free 16 KB block of expansion RAM (E-RAM) (&C4 to &FF), if one has been found. Or...
If register A contains the value 0 and the Zero flag is set, then NO free block of E-RAM has been found.
If a free E-RAM was found (physical RAM select in A), then register HL will contain a pointer to the corresponding system variable XRAM_C4 to XRAM_FF of the E-RAM block. This XRAM_?? variable has been marked with the value &81. Due to this the E-RAM block has been booked as "used".

Manipulated: AF, BC, HL, and one of the variables XRAM_C4 to XRAM_FF.

Description: The OS function EFER is used to search a free block of 16 KB E-RAM within the first 512 KB of the E-RAM (phys. &7FC4 to &7FFF).
After calling EFER the register A will contain the physical RAM select of a free 16 KB block of E-RAM (&C4 to &FF). However, if A contains 0, and the Zero flag is set, then there was no free E-RAM block.
Aside of the physical RAM select in A the register HL will contain a pointer to the corresponding XRAM system variables. Using this pointer will allow you to define its block of E-RAM free after usage. This is done by writing &01 to the variable.
However, into the same XRAM variable the value &81 has already been written by EFER. This value marks the E-RAM as "used", so it will not be used twice or by other programs.

An example:
```
 CALL EFER        ;Find a free 16 KB block of expansion RAM (E-RAM)
 JR   Z,ERROR     ;Jump away if NO free E-RAM could be found !!!
 PUSH HL          ;Save pointer to the corresponding XRAM_?? variable

 LD  B,&7F        ;Register B = I/O address of the Gate Array
 OUT (C),A        ;Bank in E-RAM between &4000 and &7FFF for usage
... ... ...       ;Now you can work with your E-RAM block...

 POP HL           ;Restore pointer to XRAM_?? variable in HL
 LD  (HL),&01     ;Book E-RAM block as "free" after usage
...
```

Attention: The System Variables from XRAM_C4 up to XRAM_FF must contain correct values before using EFER (this is usually the case). To book a E-RAM block free, which has been previously selected by this OS function EFER, you simply write the value "1" to its corresponding variable (equals the address provided in register HL).

READ TIME AND DATE FROM A REAL TIME CLOCK

Short description: Time and Date will be read from an connected Real-Time-Clock (RTC).

Label: R_RTC

ROM-number: A

Start address: &FE47

Jump in conditions: Some kind of RTC should be connected.

Jump out conditions: Date and Time will be read to the system variables UHR_??. Their former content will be moved to the system variables AUH_??.

Manipulated: AF, BC, DE, HL the system variables UHR_00 to UHR_JAR and AUH_00 to AUH_JAR. Furthermore the RAM area from &BE02 to &BE0B.

Description: There are some sorts of Real-Time-Clocks (RTCs) for the CPC computer range, to provide actual time and date.
The OS function R_RTC checks if an RTC from Dobbertin / dxs, M4 or the SYMBiFACE II or III is connected. If yes, the time and date will be read and transferred to the system variables UHR_00 to UHR_JAR (see file #OS-VAR.ENG). Time and Date are stored in the beneficial BCD format. Before the new data is read, the former values for time and date will be transferred to system variables AUH_00 to AUH_JAR.

Attention: When using this OS function either an Dobbertin RTC should be present (or the remake from dxs), the M4 board with its RTC, the SF2 or SF3 with its RTC.

CONVERT BINARY NUMBER INTO A BCD ENCODED NUMBER

Short description: A regular binary number will be converted into an binary coded dezimal (BCD) number.

Label: BIN2BCD (or BIN2BCE, if register E = 10)

ROM-number: A

Start address: &FD34 (or &FD36)

Jump in conditions: D = binary number from 0 to 99
E = &0A (E = 10, when using label BIN2BCE)

Jump out conditions: A = BCD number (&00...&99)
E = &0A (register E contains value 10)

Manipulated: AF and DE

Description: This OS function(s) converts a binary number into an BCD encoded number. BCD numbers are more easy to be displayed on screen.
Under FutureOS f.e. the numbers for time and date are stored in BCD format. Have a look at OS Variables UHR_SEK...UHR_JAR. Also some RTCs are using the BCD format (f.e. the RTC from Dobbertin, dxs or SF2).
Example: Register D will be loaded with an binary number between 0 and 99 an then the OS function BIN2BDC will be called. After the return the register A will contain the corresponding BCD number. Furthermore the register E will contain the value &0A = 10.
If you want to use this function multiple times subsequently then you can use the label BIN2BCE beginning at the second call (because register E already does contain the value 10 as divisor).

```
LD   D,&12       ;D = &12 = 18, a binary number
CALL BIN2BCD     ;Convert Binary to BCD
LD   (XXXX),A    ;A = &18, store BCD encoded number to RAM

LD   D,&2D       ;D = 45
CALL BIN2BCE     ;Convert Binary to BCD, here label BIN2BCE usable
LD   (XXXX),A    ;A = &18, store BCD encoded number to RAM
```

Attention: Binary numbers bigger than 99 can not be displayed in BCD format. Please only use numbers between 0 and 99.
If you call this OS function using the second label (BIN2BCE) then it's 2 us more quick, but register E need to be preloaded with value 10. This is the case after you did call BIN2BCD once before.

CONVERT A BCD ENCODED NUMBER TO A BINARY NUMBER

Short description: A binary coded decimal (BCD) number will be converted into an regular binary number (one bytes).

Label: BCD2BIN

ROM-number: A

Start address: &FD4D

Jump in conditions: A = BCD number from &00 to &99

Jump out conditions: A = binary number 0-99 (= &00 to &63)

Manipulated: AF and DE

Description: This OS function converts a binary coded decimal (BCD) number into binary number - this is one byte. BCD numbers can come from sources like real time clocks and they are more easy to be displayed on screen. Under FutureOS f.e. the numbers for time and date are stored in BCD format. Have a look at OS Variables UHR_SEK ... UHR_JAR. Some RTCs can only use the BCD format (f.e. the RTC from Dobbertin or dxs).

Example: Register A will be loaded with an BCD number between &00 and &99 an then the OS function BDC2BIN will be called. After the return the register A will contain the corresponding binary number between &00 and &63 (0-99 decimal).

Here is an example source:

```
LD   A,&18       ;A = &18 a BCD number
CALL BCB2BIN     ;Convert BCD to Binary
LD   (XXXX),A    ;A = &12 = 18, store binary number to RAM
```

Attention: Values bigger than &99 can not be displayed in BCD format.
So they can't be converted to regular binary (byte) numbers.
The maximum size of the result can be 99 = &63.Please only use numbers between &00 and &99 to be converted.

WAIT UNTIL THE SYMBiFACE III IS READY TO USE

Short description: Wait until the SYMBiFACE III reports to be ready.

Label: W_SF3

ROM-number: A

Start address: &FD43

Jump in conditions: The SF3 card should be connected (see T_SF3)

Jump out conditions: The SF3 is now ready to receive a command
Register BC = &FD41 = SF3 status register
Zero-Flag set: SF3 is ready to be used

Manipulated: AF and BC

Description: This OS function waits until the SYMBiFACE III reports to be ready for usage. In this case the register A will report the value from the SF3 status register: &00 (Zero flag set) = every thing fine.

But if A = &02, then there was an error.

Furthermore the register BC will return value &FD41, this is the port address of the SF3 command and status register.

Attention: Please use this OS function only if you really have connected an SF3 to your computer. You can check this by using the OS function T_SF3 once.

The most recent version of this document can be found here:

http://www.FutureOS.de