File: ‘API-B-EN.DOC’ (61 pages) Version: July 7, year of CPC 2023

Documentation of FutureOS functions located in ROM B

The OS functions located in "ROM B" are mainly concerned with Floppy-disc (merely low level) and Hard-disc (Dobbertin HD20) devices.

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 get 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.

ROM B contains all OS functions which are needed to use floppy disc drives with the FDC765 and the CPC. Both, the internal FDC and the external FDC(Vortex devices: F1-D, F1-S, M1-S and M1-D) are supported. You can use all formats from SS 40 tracks up to DD 80 tracks. Further the B ROM contains all OS functions to manage the Dobbertin HD-20 hard disc drive. And there is support for memory management too.

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

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

READ A SECTOR ID FROM DISC

Short description: An arbitrary sector ID will be read from the actual track of the inserted floppy disc.

Labels: HOLE0ID (internal FDC) or HOLE1ID (external FDC)

ROM-number: B

Start address: &C036 (HOLE0ID) or &C03B (HOLE1ID)

Jump in conditions: D = drive-number 0...3 (second head => 4..7)

Jump out conditions: 7 bytes beginning at RAM variable FDC_RES (&B840)
&B840: Status-register 0 of the FDC
&B841: Status-register 1 of the FDC
&B842: Status-register 2 of the FDC
&B843: Actual track-number of the head 0..39..79..xx
&B844: Actual head-number 0 or 1 (if using two heads)
&B845: Actual sector-number, IMPORTANT when testing the disc-format
&B846: Actual sector-size 0, 1, 2..xx

Manipulated: AF, BC, HL and FDC

Description: Read sector ID. This OS function reads the next available sector ID of the actually inserted disc from the actual track. You have to select the drive in the D register of the Z80 CPU (D = DRIVE).
Use the Bits 0 and 1 to select: 00 = drive A, 01 = drive B and so on. Further is is possible to select the head of the drive through Bit 2. When using a single-head drive you should always set the head bit to zero. You can use the internal FDC or the external FDC (from Vortex). HOLE0ID will use the internal FDC, which controls the built-in 3 inch drive. Or you can access the external FDC using OS function HOLE1ID, which controls external floppies like the Vortex F1-D.
Both OS functions read seven bytes from the FDC and write them to RAM beginning at FDC_RES (&B840). These seven bytes equal the bytes given through the result-phase of the FDC.
You can use such OS functions for example to investigate the format of an unknown disc in the drive.

Attention: If there is NO disc in drive that will lead to corrupted data. Further the system stability can be decreased. Please test if a disc is in the drive BEFORE using one of these OS functions.
If you try to access drives which aren't physically connected, that may can influence other connected drives (when the hardware decoding of the drives is insufficient).
The FDC referred as "internal FDC" is the FDC which is built in the DDI-1 or the keyboard of the CPC664 / CPC6128 / 6128Plus.
Whereas the so called "external FDC" is an FDC contained in the Vortex controller for their drives.

SEEK A TRACK (FLOPPY DISC)

Short description: This OS function moves the head of a certain floppy disc drive over a given track.

Labels: SEEK0 (internal FDC), SEEK1 (external FDC)

ROM-number: B

Start address: &C079 (SEEK0) or &C07E (SEEK1)

Jump in conditions:
D = drive 0..3
E = new track-number
Before you use the OS function SEEK, you should provide the correct step-rate-time for the FDC.

Jump out conditions: The head of the selected floppy disc drive was moved ober the desired track.
A = Status register 0 of the FDC.
L = actual/new track-number, where the head is now located.

Manipulated: AF, BC, L, AF' and the used FDC.

Description: The OS function SEEK0 is able to move the read/write-head of a floppy disc drive (given in register D) to a certain track (given in register E). This is true for the internal FDC765.
If you want to use the extenal (Vortex) FDC765, then you have to use the OS function SEEK1.
If you need to save time, then it would be better to use one of the two OS functions SINI0 or STER0. Because the step between two tracks need some milli-seconds.

Attention: The drive must be "Ready", a disc must be inserted. And the FDC must know the correct step-rate-time for the selected drive. To set the step-rate-time please use OS functions for the FDC 0 or FDC 1.

SEND COMMAND SEEK TO A FDC

Short description: The command SEEK is issued and the OS function returns immediately. These two OS functions and the two OS functions STER0/1 together can replace the two OS functions SEEK0/1.

Label: SINI0 (internal FDC) or SINI1 (external FDC)

ROM-number: B

Start address: &C0C5 (SINI0) or &C0CA (SINI1)

Jump in conditions:
D = drive 0..3
E = new track-number
Before you use the SEEK-initialize OS function, you should provide the correct step-rate-time for the FDC.

Jump out conditions: Now the head of the selected drive is moving to the selected target track (it is still moving...).

Manipulated: AF, BC and the FDC

Description: This OS function resebles the OS function SEEK (look before). But in this case the floppy disc drive head is only SEND to its target track. When this OS function returns the head is still moving, which will take some more milliseconds. This OS function doesn't wait until the head has arrived its future target track! After using the OS function SINI the FDC can't do any other command except another SEEK command.

To end the OS function SINI correctly you have to end the SEEK of the track using one of two OS functions: STER0 or STER1.

Between the CALL of SINI and STER you can do other things, so the time between must not be wasted.

Like usual every FDC (internal or external) has its own OS function:

SINI0 uses the internal FDC and SINI1 the external FDC.

Attention: After using the SINI0/1 OS function the SEEK of the drive must be ended by using the STER0/1 OS function. Else the FDC can be used in any other way. Else please see OS function SEEK0/1

END COMMAND SEEK (FDC)

Short description: This OS function waits until all moving heads of an FDC have reached their target tracks.

Label: STER0 (internal FDC) or STER1 (external FDC)

ROM-number: B

Start address: &C0EE (STER0) or &C0F3 (STER1)

Jump in conditions: Some drive head should be moving at the moment, this could be done by using OS function SINI0/1 or however.
The selection of the FDC is done through the use label STER0 or STER1.

Jump out conditions: L = new track after the SEEK.
All heads of all drives of the FDC have reached their target tracks.

Manipulated: AF, BC, HL, AF' and the FDC

Description: This OS function is used to end an previously issued SINI0 or SINI1 command. Please use STER0 to terminate a SINI0 command and STER1 to end SINI1.

When STER returns, then the FDC can be used freely again. You can use again all FDC commands.

The time between the CALL of SINI0/1 and STER0/1 can be used by the program. Both OS functions are an complete replacement of the SEEK0/1 OS function.

Attention: Beware! If you CALL this OS function when no head is moving (that means without using SINI0/1 before) the system will be frozen!!!

READ STATUS REGISTER 3 OF THE FDC

Short description: Reads status register 3 of the FDC.

Label: HOLE_S3 (internal FDC) or HOL1_S3 (external FDC)

ROM-number: B

Start address: &C0F8 (HOLE_S3) or &C0FD (HOL1_S3)

Jump in conditions: D = number of drive 0..3

Jump out conditions: A = status register 3 of the int./ext. FDC

Manipulated: AF and BC

Description: If you want to read the status register 3 of the FDC, you can use this OS function. HOLE_S3 reads the status reg. 3 of the internal FDC and HOL1_S3 reads it from the external FDC.
You can read the status 3 for each drive seperately, therefore you have to tell the OS function the drive number (in D) from 0 up to 3.
Further you can specify the head (0 or 1) in the third Bit (Bit 2) of D. The OS function provides status register 3 in the processor register A. Where the bits have the following meaning:

UNIT SELECT bit 0 and 1: Contain the selected drive (0..3).

HEAD ADRESS bit 2: Contains the selected drive head (0 or 1).

TWO SIDE Bit 3: Single-side drives provide this bit as "1" and double-sides drives set this bit to "0". When using the 3" drive of the CPC this bit is indefinite. So better don't use this bit.

TRACK 0 bit 4: This bit is set to "1" if the head of the drive is actually located over track 0, else it is set to "0".

READY bit 5: If this bit is set to "1" then the drive is READY for use. That means that a disc is inserted and the drive motor is running with the correct speed. So you can use this drive right now. If the drive is NOT READY the bit 5 is cleared to "0"

WRITE PROTECT bit 6: This bit is set to "1" if the disc is WRITE PROTECTED. But if the bit 6 is cleared to "0" then you can read AND WRITE to the inserted disc.

FAULT Bit 7: If the drive senses an error then this bit is set to "1", else the bit is cleared. But beware not all drives contain the FAULT signal.

Attention: The disc drive motors must NOT be switched on to use this OS function. You can also use HOLE_S3 without running drive motor for example to ask if a disc is inserted or not (assumed that the disc is not write protected, because this bit will be tested).

DEFINE ACCESS TIMES OF ALL DRIVES

Short description: This OS function defines the step-rate-time and the head-load-time for all drives of one FDC.

Labels: ZEIT0 (internal FDC) or ZEIT1 (external FDC)

ROM-number: B

Start address: &C12F (ZEIT0) or &C134 (ZEIT1)

Jump in conditions: D = The upper four bits contain the step-rate-time (&X0) and the lower four bits contain the head-load-time (&0X).

The head-load-time is only important for 8" drives, so you can set it to zero (then D contains a value of fromat &X0). Have a look at the step-rate-time table.

Jump out conditions: Beside the new step-rate-time and head-load-time the FDC has got a head-unload-time of 4 ms (the shortest possible, the CPC doesn't use the head-unload-time).

Further the FDC works in polling-mode (non-DMA), CPC can't use DMA.

The step-rate-time and the head-load-time (from Z80 register D) are valid for all drives of the FDC.

Manipulated: AF, BC and the access times of all drives of the FDC.

Description: Before you can use a defined drive you have to tell the FDC once how "fast" the drive is. You must provide the step-rate-time, which you have to provide in the upper four bits of register D. The lower four bits contain the head-load-time. Due to that only 8" drives use the head-load-time, you just can set it to 0.

If you choose the step-rate-time to short, then the drive is not able to move its head fast enough, an error will occur! If you choose the step-rate-time to big, then the drive will slow down. Both errors can indeed be critical. The manufacturer of a drive can provide the correct step-rate-time. For 3" drives you can work with a 12 ms step-rate. Extenal 3.5" or 5.25" drives mostly use 4 ms step-rate.

Attention: The FDC registers for the step-rate-time and so on exist only once for all connected drives (up to four are possible).

So you have to tell the FDC once the correct step-rate-time before you use a special drive. Especially when you want to use another drive you have to inform the FDC before about the step-rate-time.

3 inch and 3.5 inch as 5.25 inch drives have different step-rates.

Table of values for Z80 register D and corresponding step-rate-times:

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

The step-rate-time is valid for ALL drives connected to the FDC.

START STANDARD COMMAND PHASE - READ, WRITE, VERIFY

Short description: This collection of OS functions is used to start the command phase of the FDC. That means to set off an FDC command.
You can read/write/verify sector(s) or tracks. Errors will be reported.

Labels:	LSV0 (internal FDC) or LSV1 (external FDC). LSV0X / LSV1X correspond LSV0 / LSV1 with more trials (read, write, verify). LSV0XE / LSV1XE correspond LSV0X / LSV1X with added error-handler.

ROM-number: B

Start address:		
	&C15A (LSV0) or	&C15F (LSV1)
	&E6B2 (LSV0X) or	&E6BA (LSV1X)
	&E709 (LSV0XE) or	&E717 (LSV1XE)

Jump in conditions:
D = drive (bits 1,0) & head (bit 2) to work with.
E = track-number &00..&28..&50 (&FF), head must be over that track!
H = first sector, that should be used.
L = sector-size: 2 = 512 bytes, 3 = 1024 bytes.
Second register set of the Z80:
A' = Command code of the FDC, specifys what the FDC should do.
H' = last sector, that should be used.
L' = GAP#3, the gap between ID and data.
DE'= Source- or target-address of the data-block, that should be used.
For example the data of a sector or a complete track
The drive must be READY, the drive head must be over the right track.
LSV0X(E) / LSV1X(E): FDCLSV contains the number of read/write trials.
LSV0XE / LSV1XE: FDC_ERR and FDE_RAM point to the error-handler.

Jump out conditions: DE = pointer to byte after the end of the data-block that was used. Seven bytes beginning at FDC_RES contain the status information that the FDC has provided. Please have a look at that seven bytes!

Manipulated: AF, BC, D and AF', BC', HL' then EXX! Interrupts are OFF!
LSV0X(E) / LSV1X(E): also E, HL, DE' and FDCLSA.

Description: This OS function(s) send off the FDC command phase for every kind of data transfer. You can read, write of verify data. Since you always provide a first and an last sector you can read one or more sectors. Further you can work with complete tracks.
The content of the Z80 register A' defines what exactly the FDC will do. Before you call this OS function you have to ensure the the head of the drive is already moved over the track you want to use. The drive must report "Ready" status, that means that the READY bit must be set in the status register 3.
When using OS functions LSV0X(E) or LSV1X(E) then the RAM variable FDCLSV must contain the number of trials. FDCLSV defines how often a command is repeated if the FDC reports an error.

When using OS functions LSV0XE / LSV1XE without success (to much errors!) an error-handler is called. The error-handler must be specified through the RAM variables FDC_ERR and FDE_RAM.
When you finally call one of these OS functions the data-transfer will be done. After the data-transfer you can have a look at seven status bytes, which give information about the success of the action. They begin in RAM at variable FDC_RES. DE point to the byte after data-end.

Attention: If a SEEK command wasn't ended before you call this OS function (one of the lower four bits of the FDC main-status-register is set) the system will crash! Every SEEK command has to be terminated before, for example by using the STER0 or STER1 OS function.

FORMAT ONE TRACK IN FUTURE-OS SPECIAL FORMAT

Short description: The actual track of a disc will be formatted in the FutureOS special format.

Labels: FOR0F (internal FDC) or FOR1F (external FDC)

ROM-number: B

Start address: &C214 (FOR0F) or &C219 (FOR1F)

Jump in conditions:
D = Head (bit 2) and drive-number 0..3 (bits 1, 0)
E = Track-number (for sector ID).
The head of the drive must be located over the track, which should be formatted. The drive must report 'DRIVE READY' status. Drive-motor on!

Jump out conditions: The track is now formatted in FutureOS format.

Manipulated: AF, BC, D and L

Description: This OS function is used to format one track of a disc in the FutureOS special format. Five sectors with 1024 bytes each will be written. The following sector numbers are used: &80, &81, &82, &83 and &84. So one track provides 5 KB space.

Before you cann the OS function the head of the drive must be located over the right target track. The drive must be spinning and report the status: DRIVE READY. (The physical track number must not be equal to the track number provided in Z80 register E, but the FDC wants such a number for the sector ID. That can be used as a copy-protection.).

After you have called FOR0F or FOR1F the track under the head of the drive has been formatted in FutureOS special format with the given track number.

Attention: The drive-head must be located over the correct track, which has to be formatted - before you call the OS function. Further the drive must report "Ready". This format can be used single sided or double sided.

FORMAT ONE TRACK IN VORTEX FORMAT

Short description: The actual track of a disc will be formatted in the Vortex format.

Labels: FOR0V (internal FDC) or FOR1V (external FDC)

ROM-number: B

Start address: &C295 (FOR0V) or &C29A (FOR1V)

Jump in conditions:
D = Head (bit 2) and drive-number 0..3 (bits 1, 0)
E = Track-number (for sector ID).
The head of the drive must be located over the track, which has to be formatted. The drive must report 'DRIVE READY' status. Drive-motor on!

Jump out conditions: The track has been formatted in Vortex format.

Manipulated: AF, BC, D and L

Description: The track under the head of the drive will be formatted with nine sectors of 512 bytes each. The sectors have the following numbers: 1, 6, 2, 7, 3, 8, 4, 9 and 5. Such a track contains 4.5 KB of data. The Vortex format is double-sided format for 80 track drives.

Attention: The drive-head must be located over the correct track and the FDC must report "DRIVE READY" status.

FORMAT ONE TRACK IN DATA FORMAT

Short description: The track under the drive-head will be formatted in the Data format.

Labels: FOR0D (internal FDC) or FOR1D (external FDC)

ROM-number: B

Start address: &C315 (FOR0D) or &C31A (FOR1D)

Jump in conditions:
D = Head (bit 2) and drive-number 0..3 (bits 1, 0)
E = Track-number (for sector ID).
The head of the drive must be located over the track, which has to be formatted. The drive must report 'DRIVE READY' status. Drive-motor on!

Jump out conditions: The track has been formatted in Data format.

Manipulated: AF, BC, D and L

Description: The track under the head of the drive will be formatted with nine sectors of 512 bytes each. The sectors have the following numbers: &C1, &C6, &C2, &C7, &C3, &C8, &C4, &C9 and &C5. Such a track contains 4.5 KB of data. The Data format is used for single-sided drives with 40 tracks.

Attention: The drive-head must be located over the correct track and the FDC must report "DRIVE READY" status.

FORMAT ONE TRACK IN SYSTEM FORMAT

Short description: The track under the drive-head will be formatted in System format.

Labels: FOR0S (internal FDC) or FOR1S (external FDC)

ROM-number: B

Start address: &C395 (FOR0S) or &C39A (FOR1S)

Jump in conditions:
D = Head (bit 2) and drive-number 0..3 (bits 1, 0)
E = Track-number (for sector ID).
The head of the drive must be located over the track, which has to be formatted. The drive must report 'DRIVE READY' status. Drive-motor on!

Jump out conditions: The track has been formatted in System format.

Manipulated: AF, BC, D and L.

Description: The track under the drive-head will be formatted with nine sectors of 512 bytes. The nine sectors have the following numbers: &41, &46, &42, &47, &43, &48, &44, &49 and &45. One System track contains 4.5 KB of data. The System format is used for single-sided drives with 40 tracks.

Attention: The drive-head must be located over the correct track and the FDC must report "DRIVE READY" status (disc inserted, motor on).

FORMAT ONE TRACK IN IBM FORMAT

Short description: The track under the drive-head will be formatted in IBM format.

Labels: FOR0I (internal FDC) or FOR1I (external FDC)

ROM-number: B

Start address: &C415 (FOR0I) or &C41A (FOR1I)

Jump in conditions:
D = Head (bit 2) and drive-number 0..3 (bits 1, 0)
E = Track-number (for sector ID).
The head of the drive must be located over the track, which has to be formatted. The drive must report 'DRIVE READY' status. Drive-motor on!

Jump out conditions: The track has been formatted in IBM format.

Manipulated: AF, BC, D and L

Description: The track under the drive-head will be formatted with eight sectors of 512 bytes each. The eight sectors have the following numbers: &01, &05, &02, &06, &03, &07, &04 and &08. One IBM track contains 4 KB of data. The IBM format is used for single-sided drives with 40 tracks.

Attention: The drive-head must be located over the correct track and the FDC must report "DRIVE READY" status (disc inserted, motor on).

READ OR WRITE ONE TRACK FDC 0/1

Short description: Read or write a complete track in DATA, IBM, VORTEX or SYSTEM format. You can use the internal or the external FDC.

Labels:
S0SR (write System) /// L0SR (read System) each with FDC 0
S0VR (write Vortex) /// L0VR (read Vortex) each with FDC 0
S0DR (write Data) /// L0DR (read Data) each with FDC 0
S0IR (write IBM) /// L0IR (read IBM) each with FDC 0

S1SR (write System) /// L1SR (read System) each with FDC 1
S1VR (write Vortex) /// L1VR (read Vortex) each with FDC 1
S1DR (write Data) /// L1DR (read Data) each with FDC 1
S1IR (write IBM) /// L1IR (read IBM) each with FDC 1

ROM-number: B

Start addresses:
(S0SR) &C813 // (L0SR) &C819 // (S1SR) &C9CF // (L1SR) &C9D5
(S0VR) &C81F // (L0VR) &C825 // (S1VR) &C9DB // (L1VR) &C9E1
(S0DR) &C82B // (L0DR) &C831 // (S1DR) &C9E7 // (L1DR) &C9ED
(S0IR) &CB8B // (L0IR) &CB91 // (S1IR) &CD02 // (L1IR) &CD08

Jump in conditions:
DE = Address of the data block (4 KB or 4.5 KB)
REG08_0 = Track (drive-head must have been located over that track)
REG08_1 = Drive 0..3 (bits 0, 1) and drive-head (bit 2)

Jump out conditions: DE = Byte after the transferred data block.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, YL and REG_PC low

Description: This program-complex is used to read a complete track from disc or to write a complete track to disc. The internal or the external FDC can be used. You can choose one of the following formats: DATA-, IBM-, VORTEX- or SYSTEM-format.
Depending on the drive, the format and the data-transfer-direction you have to use another label to call the corresponding OS function. There are 16 labels (2 FDCs * 4 formats * 2 data-transfer-directions = 16).
After you have called the right OS function and the end of the data transfer, the register DE points to the byte after the end of the transferred data block. Therefore you can work directly with the next track.
The special advantage of these OS functions is that they all work with an interleave factor of "0". That means that you read or write the complete track in only one rotation of the disc, despite the fact that some formats use another interleave.

Attention: The head of the drive must be located over the correct track before you can use this routine(s). The drive status must be "Ready". And (like with all other disc data-transfer operations) the interrupts must be switched OFF.

LOAD DATA IN FIRST 64 KB - SYSTEM/DATA FORMAT

Short description: Loads a file direct into the actual 64 KB.

Labels: L0DS (internal FDC) or L1DS (external FDC)

ROM-number: B

Start address: &CE79 (L0DS) or &CF97 (L1DS)

Jump in conditions:
DE = Target address for data (or file) in first 64KB RAM.
HL = Pointer to start of "track-sector-table".
REG08_0 = Track that contains first sector.
REG08_1 = head (bit 2) and drive (bits 1, 0)
A SINI0/1 command must have been issued before. And a loading-table / track-sector-table must have been generated.

Jump out conditions: Data has been loaded (if drive was Ready before).

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, YL, and the RAM-variables REG08_0, REG_PC(low), FDC_RES and the FDC itself. Further (eventually) the header-buffer of foreground-programs (BC00 bis BC7F).

Description: This OS function is used to load a file or defined tracks and sectors to a target address. You can use every one of the eight possible drives. But the disc must have system or data format (other formats see below).
You shouldn't load data above &A000, because this will overwrite system variables. The target address (where the data should be loaded) is provided in register DE. And HL contains the pointer to the track-sector table (loading-table).

Construction of the track-sector-table (System or Data format):
The table begins with an track-number, followed by the number of sectors, that have to be loaded from that track. Then the single sector numbers follow.
Then the next track-number (, number of sectors, sector-number) follow and all again and again.
- If the number_of_sectors is &FF, then the complete track has to be loaded. In this case the track-number is only followed by the &FF byte, not sector numbers follow. After the &FF the next track-number will follow.
- If the track-number is &FF, then the end of the table has been reached. The end of the track-sector table is reported through the track-number &FF

Attention: VERY IMPORTANT!!! Before you use this OS function you must send off a SEEK command (using OS function SINI0 or SINI1) to the target-track. That's the same track which must be provided for this OS function through the RAM variable REG08_0. This OS function will end the SEEK command and start to load data.

LOAD DATA IN FIRST 64 KB - IBM FORMAT

Short description: Loads a file direct into the actual 64 KB.

Labels: L0IB (internal FDC) or L1IB (external FDC)

ROM-number: B

Start address: &D0B5 (L0IB) or &D1D3 (L1IB)

Jump in conditions:
DE = Target address for data (or file) in first 64KB RAM.
HL = Pointer to start of "track-sector-table".
REG08_0 = Track that contains the first sector.
REG08_1 = head (bit 2) and drive (bits 1, 0)
A SINI0/1 command must have been issued before. And a track-sector-table must have been generated.

Jump out conditions: Data has been loaded (if drive was Ready before).

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, YL, and the RAM-variables REG08_0, REG_PC(low), FDC_RES and the FDC itself. Further (eventually) the header-buffer of foreground-programs (BC00 bis BC7F).

Description: This OS function is used to load a file or defined tracks and sectors to a target address. You can use every one of the eight possible drives. But the disc must have IBM format.
You shouldn't load data above &A000, because this will overwrite system variables. The target address (where the data should be loaded) is provided in register DE. And HL contains the pointer to the track-sector table.
The track-sector table for IBM format is similar constructed to the table used for loading from data or system format (see before).

Attention: VERY IMPORTANT! Before you use this OS function you must send off a SEEK command (using OS function SINI0 or SINI1) to the target-track. That's the same track which must be provided for this OS function through the RAM variable REG08_0. This OS function will end the SEEK command and start to load data.

LOAD DATA IN FIRST 64 KB - VORTEX FORMAT

Short description: Loads a file direct into the actual 64 KB.

Label: L0AV (internal FDC) or L1AV (external FDC)

ROM-number: B

Start address: &D2F1 (L0AV) or &D432 (L1AV)

Jump in conditions:
DE = Target address for data (or file) in first 64KB RAM.
HL = Pointer to start of "track-sector-table".
REG08_0 = Track that contains the first sector.
REG08_1 = head (bit 2) and drive (bits 1, 0)
A SINI0/1 command must have been issued before. And a track-sector-table must have been generated.

Jump out conditions: Data has been loaded (if drive was Ready before).

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, YL, and the RAM-variables REG08_0, REG08_1, REG_PC(low), FDC_RES and the FDC itself.
Further (eventually) the header-buffer of foreground-programs (BC00 bis BC7F).

Description: This OS function is used to load a file or defined tracks and sectors to a target address. You can use every one of the eight possible drives. But the disc must have Vortex format. You shouldn't load data above &A000, because this will overwrite system variables. The target address (where the data should be loaded) is provided in register DE. And HL contains the pointer to the track-sector table.
BEWARE, the track-sector table for Vortex format has its special construction, due to the head bits.

Construction of the track-sector-table (Vortex format):
The table begins with an track/head-number, followed by the number of sectors, that have to be loaded from that track. Then the single sector numbers follow.
The track/head-number contains the head in bit 0, that means:
track/head-number = track * 2 + head
Then the next track/head-number (, number of sectors, sector-number) follow and all again and again.
- If the number_of_sectors is &FF, then the complete track has to be loaded. In this case the track/head-number is only followed by the &FF byte, not sector numbers follow. After the &FF the next track/head-number will follow.
- If the track/head-number is &FF, then the end of the table has been reached. The end of the track-sector table is reported through the track-number &FF

Attention: VERY IMPORTANT!!! Before you use this OS function you must send off a SEEK command (using OS function SINI0 or SINI1) to the target-track. That's the same track which must be provided for this OS function through the RAM variable REG08_0. This OS function will end the SEEK command and start to load data.

Load Data up to 512 KB into Expansion RAM - IBM, SYSTEM, DATA Format

Short description: OS function loads data into the expansion RAM using IBM, System or Data formatted discs. File-size can be up to 512 KB.

Labels: L016 (internal FDC) or L116 (external FDC) – at &4000 in 1. E-RAM block &C4
L015 (internal FDC) or L115 (external FDC) – any address, E-RAM

ROM-number: B

Start address: &D594 (L016), &D5A0 (L015) / &D5F3 (L116), &D5FF (L115)

Jump in conditions: Valid for every entry point:
HL = Address or the track-sector-table
REG08_1 = Head (bit 2) and drive-number 0..3 (bits 1, 0)
REG16_0 = Address of the read-track-OS function of the corresponding format
- Track-sector-must have been generated before.
- SINI0/1 must have been issued before.

Valid for entry points L015 and L115:
DE = Target address at which data should be loaded (below &8000!).
AKT_RAM = &7FC4..&7FFF, first 16 KB block of target expansion RAM, this block must be switched in before.

Jump out conditions: Data/File has been loaded into expansion RAM.
Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, and the RAM-variables: AKT_RAM, REG08_0, FDC_RES, the FDC itself and the RAM configuration.
The memory between &8000 and &8FFF could have been altered (could have been used as track-buffer).

Description: These OS functions load data from disc to the expansion RAM of the CPC. Data is written to the expansion RAM ascending 16 KB block after 16 KB block. If you have connected 512 KB expansion RAM, then you can use it all. (Although a data disc has only 178 KB space).
The expansion RAM is mapped in between &4000 and &7FFF in 16 KB blocks.
You can use all eight drives and discs with IBM, System or Data fomat.
If you use the OS functions L016 or L116 the data will be loaded at address &4000 of the first expansion RAM block &C4.
If you use the OS functions L015 or L115 then you can choose the start-address and the first RAM block. The start address must be provided in register DE, it must be below &8000. The first expansion RAM block must be written to the RAM-variable AKT_RAM and must have been switched of manually. An example:

```
LD   BC,&7FC6     ;&7F = GATE ARRAY, &C6 = third 16 KB block (1 of 32)
OUT  (C),C        ;switch 16 KB block on, between &4000 and &7FFF
LD   (AKT_RAM),BC ;write actual RAM-configuration to RAM-variable.
```

To switch on the 16 KB expansion RAM blocks ascending, you have to send the following byte to port &7Fxx, where xx can be:

`&C4,&C5,&C6,&C7 , &CC,&CD,&CE,&CF , &D4,&D5,&D6,&D7 , &DC,&DD,&DE,&DF,`

`&E4,&E5,&E6,&E7 , &EC,&ED,&EE,&EF , &F4,&F5,&F6,&F7 , &FC,&FD,&FE,&FF.`

Attention: A SINI0/1 command to the first track of the track-sector-table must have been issued before you can call one of these OS functions. These OS functions will end the SEEK command. Further the drive must report DRIVE READY before you call one of these OS functions.

LOAD DATA UP TO 512 KB INTO EXPANSION RAM - VORTEX FORMAT

Short description: OS function loads data into the expansion RAM using a Vortex formatted discs. File-size can be up to 512 KB.

Labels: L0V6 (internal FDC) or L1V6 (external FDC) - at &4000 in 1. E-RAM block &C4
L0V5 (internal FDC) or L1V5 (external FDC) - any address, E-RAM

ROM-number: B

Start address: &D652 (L0V6), &D65E (L0V5) / &D6BE (L1V6), &D6CA (L1V5)
Jump in conditions: Needed for every entry point:
HL = Address or the track-sector-table
REG08_0 = Track, to which the SINI command has been issued
REG08_1 = Head (bit 2) and drive-number 0..3 (bits 1, 0)
REG16_0 = Address of the read-track-OS function of the corresponding format
- Track-sector-must have been generated before.
- SINI0/1 must have been issued before.

Valid for entry points L0V5 and L1V5:
DE = Target address at which data should be loaded (below &8000!).
AKT_RAM = &7FC4..&7FFF, first 16 KB block of target expansion RAM, this block must be switched in before.

Jump out conditions: Data/File has been loaded into expansion RAM.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, and the RAM-variables: AKT_RAM, REG08_0, REG08_1, FDC_RES, the FDC itself and the RAM configuration.
The memory between &8000 and &8FFF could have been altered (could have been used as track-buffer).

Description: These OS functions load data from disc to the expansion RAM of the CPC. Data is written to the expansion RAM ascending 16 KB block after 16 KB block. If you have connected 512 KB expansion RAM, then you can use it all. (Although a data disc has only 178 KB space).
The expansion RAM is banked in from &4000 to &7FFF in 16 KB blocks.
You can use all eight drives and discs with Vortex fomat.
If you use the OS functions L0V6 or L1V6 the data will be loaded at address &4000 of the first expansion RAM block &C4.
If you use the OS functions L0V5 or L1V5 then you can choose the start-address and the first RAM block. The start address must be provided in register DE, it must be below &8000. The first expansion RAM block must be written to the RAM-variable AKT_RAM and must have been switched of manually. An example:

```
LD   BC,&7FC5     ;&7F = GATE ARRAY, &C5 = second 16KB block (1 of 32)
OUT  (C),C        ;switch 16 KB block on, between &4000 and &7FFF
LD   (AKT_RAM),BC ;write actual RAM-configuration to RAM-variable.
```

To switch on the 16 KB expansion RAM blocks ascending, you have to send the following byte to port &7Fxx, where xx can be:

`&C4,&C5,&C6,&C7 , &CC,&CD,&CE,&CF , &D4,&D5,&D6,&D7 , &DC,&DD,&DE,&DF, &E4,&E5,&E6,&E7 , &EC,&ED,&EE,&EF , &F4,&F5,&F6,&F7 , &FC,&FD,&FE,&FF`.

Attention: A SINI0/1 command to the first track of the track-sector-table must have been issued before you can call one of these OS functions. These OS functions will end the SEEK command. Further the drive must report DRIVE READY before you call one of these OS functions.

LOAD DIRECTORY OF A FLOPPY DISC DRIVE

Short description: Loads the directory of a drive to the DIR-buffer.

Label: LESEDIR (internal FDC) or LES1DIR (external FDC)

ROM-number: B

Start address: &FDFA (LESEDIR) or &FDFD (LES1DIR)

Jump in conditions: YL = Drive 0..3 (bits 1, 0)
The system variable TURBO_X must contaion correct values.

Jump out conditions: The DIRectory has been read to a block below the address specified through TURBO_X. Further the RAM variable TURBO_X was corrected to the new situation. REG08_0 contains the drive-number 0..7 (A,B,..,H) and A contains the length of the DIRectory in 256 byte pages (DIR lenght/256). The DIR is actually not sorted.

Manipulated: AF, BC, DE, HL, AF', BC', HL', IX, YH and the RAM-variables REG08_0, REG16_0..7, FDC_RES, the FDC itself and the RAM configuration.

Description: FutureOS buffers all DIRectories in (expansion) RAM, that makes the system faster. But you have to keep in mind that you have to read the DIRs again after changing a disc.

Both OS functions are used to read the DIR of one drive into the DIR-buffer of the (expansion) RAM.

Discs can have Data, System, IBM or Vortex Format.

The Z80 register YL must contain the number of the drive, from which the DIR should be read (0..3). The selection of the internal or external FDC is done by using the correct Label/entry point.

The RAM variable TURBO_X (managed by the OS!) defines to which address the DIR will be read. The DIR will be read below the given upper limit, then variable TURBO_X will be adapted (all done by OS).

If there is no space in the expansion RAM then the DIR will be read in the first 64 KB of memory.

Attention: The read DIRectory will NOT be sorted automatically.

LOAD DIRECTORY OF EVERY TAGGED DRIVE

Short description: Loads the directories of all tagged drives into the expansion RAM directory buffer and sorts them.

Label: LESDIR1

ROM-number: B

Start address: &FDEB

Jump in conditions: The system variables TURBO_A..M and TURBO_X must contain correct values (normally done by the OS).
REG08_0 should contain &FF, else the DIRectory of the corresponding drive is seen as previously loaded and unsorted (will be sorted).

Jump out conditions: The directories of all tagged drives have been loaded and sorted. System variables have been adapted.

Manipulated: AF, BC, DE, HL, AF', BC', HL', IX, IY and the RAM-variables REG08_0, REG16_0..7, TURBO_A..M, TURBO_X, FDC_RES, the FDC itself and the RAM configuration.

Description: This entry point is used to read all the directories of all tagged drives (A..H) into the directory buffer (expansion RAM).

While the drives step their heads the directories will be sorted, so you an use them after the return of the OS function.

If one of the tagged drives is NOT ready the OS function don't returen. In this case the drive will be marked as untagged and the OS function jumps in the Desktop of FutureOS.

The discs of the used drives can have System, IBM or Vortex format.

Attention: The loaded directories will be sorted. If an read-error occurs the OS function jumps to the Desktop, which displays an error message.

CALCULATION OF A 67 BYTES CHECKSUM OF A AMSDOS/FUTUREOS FILE

Short description: The checksum of a file-header (AmsDOS or FutureOS) will be calculated.

Label: TST_HED

ROM-number: B

Start address: &D75B

Jump in conditions: DE = Start-address of the 128 byte header-record.

Jump out conditions: HL = Checksum of the header (just calculated)
DE = Pointer to the low byte of the (potential) checksum of the header record of the file.
B = &00

Manipulated: AF, BC, DE, HL and XL

Description: All AmsDOS files (except ASCII files) have a so called file-header, this header is part of the 128 byte header record. The user won't see this header record under AmsDOS.

Also FutureOS files can contain such a header. FutureOS uses the first 128 bytes of a file as header too, but this header is strongly expanded (its still compatible to AmsDOS).

The AmsDOS (and also FutureOS) generate an checksum over the first 67 bytes (0-66). The checksum is then written into bytes 67 and 68 of the header.

If you want to know if a file has a header, just use this OS function to calculate the checksum and compare it with bytes 67 and 68 of the assumed header. If both 16 bit values are equal the corresponding file has indeed a file-header of 128 bytes.

Attention: If the first record of a file conains only zero bytes, than this can cause problems. In this case it's not clear if it is an header or not (what is most often the case).
Please read in the big FutureOS handbook if you want to know how a AmsDOS or FutureOS header look like.

WAIT UNTIL DRIVE IS READY (MAX. 5 SECONDS)

Short description: Test if a defined drive reports READY. If not, wait until it is READY. But there is a maximum waiting time.

Labels:	LWR0 (internal FDC) or LWR1 (external FDC) - wait up to 5. sec LR0S (internal FDC) or LR1S (external FDC) - any waiting time

ROM-number: B

Start address: &DC49 (LWR0), &DC59 (LWR1) / &DC4C (LR0S), &DC5C (LR1S)

Jump in conditions:
Valid for all entry points:
D = number of drive (0..3)
Further valid for OS functions LR0S and LR1S:
HL = Number of tries. How often the READY-state should be tested

Jump out conditions:
Zero-flag = 0 (cleared) => drive is READY and A = FDC status reg. 3
Zero-flag = 1 (set) => drive is NOT ready

Manipulated: AF, BC and HL

Description: This OS function sense the status of the drive. If you want to work with a drive this drive must report READY status. That means that the drive is ready to transfer data.

When calling this OS function Z80 register D must contain the number of the drive (0..3). Depending on the FDC you use (internal/external) you will use another label to call the OS function.

If you use the label LWR0 (internal FDC) or LWR1 (external FDC) the maximal waiting-time is five seconds, then the OS function returns, even the drive is still NOT ready (the Zero flag is set). But when the OS function returns with cleared zero-flag, then the drive is READY for a data transfer.

Well, if you don't want to wait up to 5 seconds for a response, then you can use the labels LR0S and LR1S. In this case the Z80 register HL must contain the number of tries (how often the OS function tries to get a READY signal from the drive). A number-of-tries of &8000 (32768) equals a waiting time of about five seconds.

Attention: If you don't want to use a obligatory run-up time for the floppy disc drive, then you can use this OS function to get faster informaion if the drive is already READY. Because this OS function returns as soon as the drive IS READY and don't wait a fixed time-period. No time is wasted. This is a big advantage if you have much drive access.
The FutureOS doesn't use run-up / head load times at all.

Wait until DRIVE is READY (max. 5 Seconds) - ABORT at WRITE Protection

Short description: Test if a defined drive reports READY. If not, wait until it is READY. But there is a maximum waiting time. If the disc is write protected an error message will be provided.

Labels: LWS0 (internal FDC) or LWS1 (external FDC) - wait up to 5 sec.
LS0S (internal FDC) or LS1S (external FDC) - any waiting time

ROM-number: B

Start address: &DC69 (LWS0), &DC7D (LWS1) / &DC6C (LS0S), &DC80 (LS1S)

Jump in conditions:
Valid for all entry points:
D = number of drive (0...3)
Further valid for OS functions LS0S and LS1S:
HL = Number of tries. How often the READY-state should be tested

Jump out conditions:
Zero-flag = 0 (cleared) => drive is READY and A = FDC status reg. 3
Zero-flag = 1 (set) => drive is NOT ready
If the disc in the drive was write-protected, an error message has been displayed and the stack pointer SP is increased by two.
Subsequently the OS function jumps back to the Desktop (using entry point FORA), in this case it doesn't return.

Manipulated: AF, BC and HL

Description: This OS function sense the status of the drive. If you want to work with a drive this drive must report READY status. That means that the drive is ready to transfer data. Further the read/write status of the disc will be investigated.

When calling this OS function Z80 register D must contain the number of the drive (0..3). Depending on the FDC you use (internal/external) you will use another label to call the OS function.

If you use the label LWS0 (internal FDC) or LWS1 (external FDC) the maximal waiting-time is five seconds, then the OS function returns, even the drive is still NOT ready (the Zero flag is set). But when the OS function returns with cleared zero-flag, then the drive is READY for a data transfer.

Well, if you don't want to wait up to 5 seconds for a response, then you can use the labels LS0S or LS1S. In this case the Z80 register HL must contain the number of tries (how often the OS function tries to get a READY signal from the drive). A number-of-tries of &8000 (32768) equals a waiting time of about five seconds.

Further this OS function makes an additional test if the insterted disc can be written. If the disc in the drive is write-protected, then the stack pointer SP will increased by two, an error-message will be shown ("Disc is write protected") and the OS function jumps to the Destop using the FORA entry point (see ROM D).

Attention: If you don't want to use a obligatory run-up time for the floppy disc drive, then you can use this OS function to get faster informaion if the drive is already READY. Because this OS function returns as soon as the drive IS READY and don't wait a fixed time-period. No time is wasted. This is a big advantage if you have much drive access.

The FutureOS doesn't use run-up times at all.

Beware! If the disc in drive is write protected then this OS function will NOT return.

RECALIBRATE DRIVE (SEARCH TRACK 0)

Short description: Recalibrate one drive. All eight drives usable.

Label: REA0 (internal FDC0), REA1 (external FDC)

ROM-number: B

Start address: &C058 (REA0), &C05D (REA1)

Jump in conditions: D = drive 0..3 (int/ext. FDC doesn't matter)
The drive must report "DRIVE READY"

Jump out conditions: The selected drive has been recalibrated, its head is now located over track zero (Look at register L!).
A = FDC status register 0.
L = Number of track where the head is now located. Should be &00.

Manipulated: AF, BC, L and AF'. The head of the drive was moved.

Description: When you switch on the CPC, the FDC doesn't know where the heads of the drives are located (over which track). Therefore you have to recalibrate the heads of all drives once after you switch on the CPC. (It makes also sense after some kinds of drive errors).

To recalibrate a drive means to move its head ober track 0 (the first track) of the disc. Every drive has a track-0-sensor, which tells the FDC that the head of a drive is located over track 0.

The FutureOS recalibrates the head of a drive just before it read the directory of a disc. If a drive is physically not connected and you try to recalibrate it, the system will stall or crash.

Attention: Beware! Never recalibrate a drive which is not connected.
Test if the drive is ready before you issue the recalibrate command.
Else the system can crash or stall.

STEP-RATE-TIME(KONFIG) EINES LW AKTIVIEREN

Short description: The corresponding step-rate-time of a drive (given by the OS) will be activated.

Labels: SWZ0 (internal FDC) or SWZ1 (external FDC)

ROM-number: B

Start address: &C129 (SWZ0), &C121 (SWZ1)

Jump in conditions: A = drive 0..3

Jump out conditions: The step-rate-time (Step Rate) of the selected drive has been activated.

Manipulated: AF, BC, D and HL. The step-rate-time was changed.

Description: The ROM B of FutureOS contains a value for the step-rate-time for every single floppy disc drive. The step-rate-time can be configured by the user for every drive. Using this OS function allows to prepare the FDC with the step-rate-time of a defined drive.

An application should always use this OS function when changing the drive, because other drives may have bigger or smaller step-rate-times predefined.

The usage of different step-rate-times allows to operate ever floppy disc drive the fastest possible way.

Examples: For the internal 3" drive the step-rate-time is 12 ms, while it's only 4 ms for external 3.5" drives.

Attention: Please use this OS function every time when switching the drive. Else the FDC would use the wrong step-rate-time.

READ 7 BYTES OF THE RESULT PHASE OF A FDC

Short description: Reads the seven bytes of the result phase from FDC and writes them into RAM.

Label: FDC0_RE (both FDCs)

ROM-number: B

Start address: &C1C4

Jump in conditions: BC = FDC data register (internal or external FDC)
The FDC must report "Result Phase". 7 bytes must be ready to be read.

Jump out conditions: 7 Bytes were written into RAM, beginnin at FDC_RES.

Manipulated: AF, C, HL, the FDC and 7 bytes RAM beginning at FDC_RES.

Description: If the FDC has to execute an command with result-phase, this OS function can be used to get the seven result-bytes and write them into RAM. Normally this is done by the OS itself. But this OS function can be helpfull if you want to program the FDC directly.

Before you call this OS function you have to load register BC with the port of the FDC data register, this also does the FDC selection.

FDC0_RE reads the seven result bytes from the FDC and writes then into RAM beginning at RAM variable FDC_RES.

If you want the result bytes at another address in RAM, just jump to the address of this OS function plus three bytes and load HL with the target address before.

Attention: If the FDC doesn't report "Result phase" before you call FDC0_RE, this can cause errors or even stall or crash the system.

GET THE STEP-RATE-TIMES OF ALL DRIVES FROM ROM

Short description: Copies the standard step-rate-times of all eight drives from ROM B to system RAM and activates them.

Label: SWZGEN

ROM-number: B

Start address: &C01F

Jump in conditions: -

Jump out conditions: Standard step-rate-times from ROM B activated.

Manipulated: BC, DE, HL and all eight step-rate-times in RAM.

Description: ROM B contains the step-rate-times of all eight drives beginning at RSWZ. Originally they were defined by the user of this version of FutureOS.

SWZGEN copies the standart-step-rate-times (SWZ) of all eight drives from ROM B into the system RAM. Then they're activated, because all FutureOS disc-OS functions use the step-rate-times of the system RAM.

It makes sense to call SWZGEN after something has altered the step-rate-times in the system RAM and an error has occured or you didn't need the alterations any longer.

Attention: The old step-rate-times of the RAM have been overwritten.
Therefore the step-rate-times of FutureOS ROM B should make sense.

FROMAT DISC (DATA, SYSTEM, IBM or FutureOS)

Short description: These OS functions format a disc with Data, System, FutureOS or IBM format. You can use the internal or the external FDC.

Labels:
Format from track &00(0) up to &28(40):
F0DAT (Data format, internal FDC) F1DAT (Data format, exter.FDC)
F0SAT (System format, internal FDC) F1SAT (System format, ext.FDC)
F0FAT (FutureOS format, internal FDC) F1FAT (FutureOS form, ext.FDC)
F0IAT (IBM format, internal FDC) F1IAT (IBM format, extern.FDC)

Format with free selection of first and last track:
F0DAU (Data format, internal FDC) F1DAU (Data format, exter.FDC)
F0SAU (System format, internal FDC) F1SAU (System format, ext.FDC)
F0FAU (FutureOS format, internal FDC) F1FAU (FutureOS form, ext.FDC)
F0IAU (IBM format, internal FDC) F1IAU (IBM format, extern.FDC)

ROM-number: B

Start address:
&C4B7 (F0DAT) // &C518 (F1DAT) // &C579 (F0SAT) // &C5CD (F1SAT)
&C621 (F0FAT) // &C675 (F1FAT) // &C76B (F0IAT) // &C7BF (F1IAT)
&C4BC (F0DAU) // &C51D (F1DAU) // &C57E (F0SAU) // &C5D2 (F1SAU)
&C626 (F0FAU) // &C67A (F1FAU) // &C770 (F0IAU) // &C7C4 (F1IAU)

Jump in conditions: Valid for all entry points:
D = Head 0/1(bit 2) and drive-number 0..3(bits 1, 0)
YH = &00 => Normal formatting (for example 40 track drives)
YH = &FF => Formatting with double-step, 40 trk.format on 80 trk. drv

For entry points F0DAU, F1DAU, F0SAU, ... ,F1IAU is also valid:
YL = First track (inclusive) that has to be formatted.
A = Last track (inclusive) that has to be formatted.

Further the disc-drive has to report "DRIVE READY" status.

Jump out conditions: The selected tracks has been formatted.

Manipulated: AF, BC, DE, HL, AF', IX, IY and the step-rate-time.

Description: This collection of OS functions is used to format floppy discs. You can use Data, System, FutureOS or IBM format. You can also use the internal FDC (drives A,B,C,D) or the external FDC (drives E, F, G, H). Further you can choose if you want to format with double steps or not. By using the right label you can choose if you want to use the standard tracks (0 up to 40) or if you like to choose freely which tracks should be formatted.

Attention: The selected disc in drive must be writeable. All data of the inserted disc will be lost after the disc has been formatted.

FORMAT DISC IN VORTEX FORMAT

Short description: Format a disc with Vortex format using the internal or the external FDC.

Labels:
Format from track &00(0) up to &50(80):
F0VAT (internal FDC) // F1VAT (external FDC)

Format with free selection of first and last track:
F0VAU (internal FDC) // F1VAU (external FDC)

ROM-number: B

Start address: &C6C9 (F0VAT) / &C71A (F1VAT) // &C6CE (F0VAU) / &C71F (F1VAU)

Jump in conditions:
D = Drive 0..3
F0VAU, F1VAU further needs:
YL = First track (inclusive) that should be formatted.
A = Last track Spur (inclusive) that should be formatted.

Further the disc-drive has to report "DRIVE READY" status.

Jump out conditions: The selected tracks has been formatted.

Manipulated: AF, BC, DE, HL, AF', IX, IY, FDC and the step-rate-time

Description: These OS functions allow to format a disc in Vortex format. You can use the internal FDC (drives A,B,C,D) or the external FDC (drives E, F, G, H).

Either you use the standard tracks from 0 to 80, or you choose the first and last tack freely. The selection is done by using different labels.

Attention: The selected disc in drive must be writeable. All data of the inserted disc will be lost after the disc has been formatted.

SAVE UP TO 64 KB OF DATA (DATA, SYSTEM, IBM FORMAT)

Short description: Up to 64 KB of data will be saved to a disc in Data, System or IBM format. But NO directory entry is made

Labels: S0DS (internal FDC) or S1DS (external FDC)

ROM-number: B

Start address: &D771 (S0DS) // &D7CC (S1DS)

Jump in conditions:
The save-table must have been generated and the SEEK command must have been issued.
HL = Start address of the save-table
DE = Start address of data to be saved to disc
REG08_0 = First track, where data should be saved
REG08_1 = Head 0/1(bit 2) and drive 0..3(bits 1, 0)
REG16_0 = Address of track-save OS function (see below)

Jump out conditions: The source data has been saved on disc to the tracks and sectors defined through the save-table.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, YL, and the RAM-variables REG08_0, REG_PC(low), FDC_RES and the FDC.

Description: This OS function allows to save any data from the actual 64 KB of RAM to disc. Discs with Data, System or IBM format can be used. The desired disc format is defined through RAM variable REG16_0.

It contains the address of the track-save-OS function, which depends on the used format and FDC (Please look at "READ OR WRITE ONE TRACK FDC0/1"). This OS function use a so called save-table, its construction is the same as a track-sector-table for loading files (see before) from discs with Data, System or IBM format.

Before you can call this OS function you have to issue a SEEK command to the first track of the save-table (use SINI0/1). Don't terminate the SEEK command, that is done by this OS function.

When the save-data process was successful the source data of RAM was written to the tracks and sectors previously defined through the save-table. But NO entry was made for the directory.

Attention: NO entry is made in the directory of the disc.

SAVE UP TO 64 KB OF DATA (VORTEX FORMAT)

Short description: Up to 64 KB of data can be saved to a disc with Vortex forma. But NO directory entry is made

Labels: S0AV (internal FDC) or S1AV (external FDC)

ROM-number: B

Start address: &D826 (S0AV) // &D884 (S1AV)

Jump in conditions: The save-table must have been generated and the
SEEK command must have been issued.
HL = Start address of the save-table
DE = Start address of the data to be saved to disc
REG08_0 = First track, where data should be saved
REG08_1 = Head 0/1(bit 2) and drive 0..3(bits 1, 0)

Jump out conditions: The source data has been saved on disc to the tracks and sectors defined through the save-table.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, YL, and the RAM-variables REG08_0, REG08_1, REG_PC(low), FDC_RES and the FDC.

Description: This OS function allows to save any data from the actual 64 KB of RAM to disc. Discs must have Vortex format.

This OS function use a so called save-table, its construction is the same as a track-sector-table for loading files (see before) from discs with Vortex format.

Before you can call this OS function you have to issue a SEEK command to the first track of the save-table (use SINI0/1). Don't terminate the SEEK command, that is done by this OS function.

When the save-data process was successful the source data of RAM was written to the tracks and sectors previously defined through the save-table. But NO entry was made for the directory.

Attention: NO entry is made in the directory of the disc.

SAVE DATA UP TO 512 KB (DATA, SYSTEM, IBM FORMAT)

Short description: Data from the expansion RAM up to 512 KB will be saved to a floppy disc in Data, System or IBM format.

Labels:
S016 (internal FDC, from &4000 in ERAM &C4) // S015 (internal FDC, free Start address)
S116 (external FDC, from &4000 in ERAM &C4) // S115 (external FDC, free Start address)

ROM-number: B

Start address: &D8DD (S016) // &D8E9 (S015) // &D9AF (S116) // &D9BB (S115)

Jump in conditions:
For all entry points valid:
The save-table must have been generated and the SEEK command must have been issued(using SINI).
HL = Start address of the save-table
YH = Track length/256, that means &10 for IBM or &12 for Data, System
REG08_1 = Head 0/1(bit 2) and drive 0..3(bits 1, 0) to save the data
REG16_0 = Address of the track-save OS function

The labels S015 and S115 further need the following data:
DE = Start address of the source-data between &0000 and &7E00 (normally between &4000...&7E00)
AKT_RAM = First 16 KB expansion RAM block or source data
This block (&C4, &C5,.., &FF) already must be switched in.

Jump out conditions: Data has been written to the disc.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, YL, and the RAM-variables AKT_RAM, REG08_0, REG_PC(low), FDC_RES and the FDC, further the track-sector-overrun-buffer between &8000 and &8FFF.

Description: This OS function is used to write data from the expansion RAM to a disc. Data-, System- or IBM-format can be used.
If you use the label S016 or S116 as entry points, the E-RAM will be saved beginning at address &4000 of E-RAM block &C4 (this is the first E-RAM block).
If you use the entry points S015 or S115 you can freely choose the start address and start block.
In every case the expansion RAM will be saved 16 KB block after block.
But no entry is made to the directory of the disc.

Attention: NO entry is made in the directory of the disc.

SAVE DATA UP TO 512 KB (VORTEX)

Short description: Data from the expansion RAM up to 512 KB will be saved to a floppy disc in Vortex format.

Labels:
S0V6 (internal FDC, from &4000 in ERAM &C4) // S0V5 (internal FDC, free start address)
S1V6 (external FDC, from &4000 in ERAM &C4) // S1V5 (external FDC, free start address)

ROM-number: B

Start address: &DA81 (S0V6) // &DA8D (S0V5) // &DB61 (S1V6) // &DB6D (S1V5)

Jump in conditions:
For all entry points valid:
The save-table must have been generated and the SEEK command must have been issued (using OS function SINI).
HL = Start address of the save-table
REG08_0 = First track, to that should be saved 0..79.
REG08_1 = Head 0/1(bit 2) and drive 0..3(bits 1, 0) to save the data

The labels S0V5 and S1V5 further need the following data:
DE = Start address of the source-data between &0000 and &7E00 (normally between &4000...&7E00)
AKT_RAM = First 16 KB expansion RAM block or source data
This block (&C4, &C5,.., &FF) already must be switched in.

Jump out conditions: Data has been written to the disc.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX, YL and the RAM-variables AKT_RAM, REG08_0, REG08_1, REG_PC(low), FDC_RES and the FDC, as the track-sector-buffer between &8000 and &8FFF.

Description: This OS function is used to write data from the expansion RAM to a disc with Vortex format.
If you use the label S0V6 or S1V6 as entry points, the E-RAM will be saved beginning at address &4000 of E-RAM block &C4 (this is the first E-RAM block).
If you use the entry points S0V5 or S1V5 you can freely choose the start address and the start block.
In every case the expansion RAM will be saved 16 KB block after block.
But no entry is made to the directory of the disc.

Attention: NO entry is made in the directory of the disc.

WRITE THE DIRECTORY OF ONE DRIVE TO DISC

Short description: Writes the DIRectory from the DIR-buffer to the corresponding disc.

Labels:
XSRIN0 (internal FDC, wait) // SRIN0 (internal FDC, OHNE don't wait)
XSRIN1 (external FDC, wait) // SRIN1 (external FDC, OHNE don't wait)

ROM-number: B

Start address: &FDF7 (XSRIN0) // &FDF4 (SRIN0) // &FDF1 (XSRIN1) // &FDEE (SRIN1)

Jump in conditions: D = Drive 0...3 (bits 1, 0)
The system variables of all drives must contain correct values.

Jump out conditions: YL = Drive 0..3 (bits 1, 0)

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

Description: This OS function is used to write back a DIRectory from the expansion RAM buffer to the floppy disc.
After using this OS function you have to wait the time, that the FDC needs to write the GAP#3 bytes. This is needed when the head of the drive should be moved after wrting the DIR to the disc.
XSRIN0 or XSRIN1 automatically wait the GAP#3 time. These OS functions return after the waiting-time.
But if you use the entry points SRIN0 or SRIN1 you have to wait the correct time by an own OS function, IF NEEDED. You can use HOLE_ID to wait until the GAP#3 time is over.
But you only have to wait the GAP#3 time if you want to use the same disc directly again, after writing the DIR.
Thinking about disc format informations etc. the system variables will be used.

Attention: If you use SRIN0 or SRIN1 and you would like to use the same drive immediately again, you MUST wait the GAP#3 writing-time.
This can be done by reading the next sector ID (by using HOLE_ID).

SAVE ALL CHANGED DIRECTORYS OF TAGGED DRIVES BACK TO DISC

Short description: All the directories of tagged drives will be written from the DIRectory buffer back to disc.

Label: SIDIR

ROM-number: B

Start address: &FDE8

Jump in conditions: System-variables of all drives, HD, MD correct.

Jump out conditions: The DIRs of all previously tagged drives have been written back to its discs.

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

Description: Every floppy disc drive, every hard-disc partition and the memory drive have their own system-variables which report if a DIR has been read and if a DIR has been changed.
If a previously read DIR has been changed, this DIRectory must be written back (from expansion RAM buffer) to the corresponding disc.
This OS function takes a look at the system variables of every device (A-M).
If a DIRectory has been previously read AND has been altered, then it will be written back to disc.

Attention: Only the DIRs will be saved that have been altered.

GENERATE TABLE OF PHYSICAL EXPANSION RAM BLOCKS

Short description: Generate a table, consisting of free 16 KB E-RAM blocks of the expansion RAM. The table is physically sorted.

Label: GTPRB

ROM-number: B

Start address: &FDE5

Jump in conditions: HL = Target address of table, and HL + 34 < &??00

Jump out conditions:
The table has been written to RAM (HL) beginning with a &00 byte (lower end of table).
A = &FD
BC = &7FFF
DE = &B9EF = XRAM_FF
HL = Upper end of table in RAM

Manipulated: AF, BC, DE and L

Description: If some free 16 KB expansion RAM blocks are needed, this OS function investigate the free RAM and generate a table of free 16 KB RAM blocks.

First GTPRB tests the block &C4, then &C5 and so on, it end at block &FF. The generated table is constructed that way:

- It begins with the byte &00 (that means you can read it also backwards). The null byte indicates the lower end of the table. Remember: After using GTPRB HL points to the upper end of table.

- The first byte 0 is followed by the numbers of all free 16 KB exp. RAM blocks (these numbers are physical E-RAM selects &C4-&FF). That numbers are sorted upwards.

Attention: The start of the table is defined by byte &00, but the end of the table is only reported in register HL.
If you call GTPRB the low-byte of HL must be fewer then &DE, because only L is increased (not H). So the table must be located within one page of &100/256 bytes.

READ ONE TRACK OF 8.5 KB - HD20

Short description: Reads one track of 8.5 KB from the Dobbertin HD20 hard disc.

Label: HDLSPUR or HDLSP (little faster).

ROM-number: B

Start address: &DCC6 (HDLSPUR), &DCCC (HDLSP)

Jump in conditions:
HDLSPUR:
B = Head-number from 0 to 3
DE = Cylinder-number from 0 to 613
HL = Target-address for 8.5 KB (= &2200) of data

HDLSP:
D = CC000000 = 2 bit MSB cylinder.
E = LSB cylinder, lower 8 bit of the cylinder-number from 0 to 613.
HL = Target address for 8.5 KB data.
XL = Head-number from 0 to 3.

Jump out conditions:
A = &00 => Success, data has been read, HL = HL + &2200
A = &02 => The sector was not found.
A not equal to &00: An error has occured, HL is undefined.
If the track was read without an error the Accu is set to &00 and register HL was increased by 8.5 KB (&2200). So HL points to byte after the end of the read data.

Manipulated: AF, BC, D, HL and IX

Description: This OS function reads a complete track of 8.5 KB length from the Dobbertin HD20. There are two enty points HDLSPUR and HDLSP.
The difference between them is in which Z80 registers the data has to be loaded before you call the OS function (look at Jump in conditions).
HDLSPUR converts the provided data of the Z80 registers to another format and provides them for HDLSP.
This OS function reads all sectors of an hard-disc track exactly in the fastest sequence they can be read. If you want to read a bigger file from hard-disc you will often have to read complete tracks. Thats for what this OS function is good for.
If you would read it all sector after sector the speed would decrease dramatically.
If you use the direct entry point HDLSP then you have to split the 10 bit cylinder number (0..613) to two registers. The lower eight bit must be provided in register E. The upper two bits must be written in the upper two bits of register D. The lower six bits of register D must be set to zero.

Attention: The hard disc must report ready. If the hard disc is not ready and you try to use this OS function, this can lead to unknown bad errors.
If you use HDLSP the lower six bits of Z80 register D must be set to zero, else the track will not be read complete.

WRITE ONE TRACK OF 8.5 KB - HD20

Short description: Writes one track of 8.5 KB from the Dobbertin HD20 hard disc.

Label: HDSSPUR or HDSSP (little faster).

ROM-number: B

Start address: &DF09 (HDSSPUR), &DF0F (HDSSP)

Jump in conditions:
HDSSPUR:
B = Number of head from 0 to 3
DE = Number of cylinder from 0 to 613
HL = Source-address of the 8.5 KB (= &2200) data

HDSSP:
D = CC000000 = upper 2 bit MSB cylinder.
E = LSB cylinder, lower 8 bit cylinder from 0 to 613.
HL = Source address of 8.5 KB data.
XL = Number of head from 0 to 3.

Jump out conditions:
A = &00 => Success, data has been written, HL = HL + &2200
A = &02 => The sector was not found.
A not equal to &00: An error has occured, HL is undefined.
It the track was written without an error the accu is set to &00 and register HL was increased by 8.5 KB (&2200). So HL points to byte after the end of the written data.

Manipulated: AF, BC, D, HL and IX

Description: This OS function writes a complete track of 8.5 KB length to the Dobbertin HD20. There are two enty points HDSSPUR and HDSSP.
The difference between them is in which Z80 registers the data has to be loaded before you call the OS function (look at Jump in conditions).
HDLSPUR converts the provided data of the Z80 registers to another format and provides them for HDSSP.
This OS function writes all sectors of an hard-disc track exactly in the fastest sequence they can be written. If you want to write a bigger file from hard-disc you will often have to write complete tracks. Thats for what this OS function is good for.
If you would write it all sector after sector the speed would decrease dramatically.
If you use the direct entry point HDSSP then you have to split the 10 bit cylinder number (0..613) to two registers. The lower eight bit must be provided in register E. The upper two bits must be written in the upper two bits of register D. The lower six bits of register D must be set to zero.

Attention: The hard disc must report ready. If the hard disc is not ready and you try to use this OS function, this can lead to unknown bad errors.
If you use HDSSP the lower six bits of Z80 register D must be set to zero, else the track will not be read complete.

READ ONE 512 BYTE SECTOR - HD20

Short description: Reads one sector of 512 bytes from the Dobbertin HD20 hard disc.

Label: HDLESE or HDLES (little faster).

ROM-number: B

Start address: &DCF8 (HDLESE), &DCFF (HDLES)

Jump in conditions:

HDLESE:
B = Number of head from 0 to 3.
C = Number of sector from 0 to 16 // &00 to &10.
DE = Number of cylinder from 0 to 613.
HL = Target address of 0.5 KB data.

HDLES:
D = CCSSSSSS = 2 bit MSB cylinder, the lower 6 bit contain the sector number from 0 to 16.
E = LSB cylinder, lower 8 bit of the cylinder-number from 0 to 613.
HL = Target address of 0.5 KB data.
XL = Number of head from 0 to 3.

Jump out conditions:

A = &00 => Success! Data has been read. HL = HL + &0200. Zero flag set
A = &02 => Sector was not found
A unequal &00: An error has occurred
It the sector was read without an error the Z80 register A is set to &00 and register HL was increased by 0.5 KB (&0200). So HL points to byte after the end of the read data.

Manipulated: AF, BC, and HL. Using HDLESE further: D and XL

Description: This OS function reads one sector of 512 bytes from the Dobbertin HD20 hard-disc. You can use two entry points: HDLESE or HDLES. The difference between them is in which Z80 registers the data has to be loaded before you call the OS function (look at Jump in conditions).
HDLESE converts the provided data of the Z80 registers to another format and provides them for HDLES. If you want to read all sectors of an track you should use HDLSPUR or HDLSP.
If you use the direct entry point HDLES then you have to split the 10 bit cylinder number (0..613) into two registers. The lower eight bit must be provided in register E. The upper two bits must be written in the upper two bits of register D. But attention: Register D further contains in its lower six bits the sector number (from 0 to 16).

Attention: The hard disc must report ready. If it doesn't so and you try to use this OS function, this can lead to unknown errors.

WRITE ONE 512 BYTE SECTOR - HD20

Short description: Writes one sector of 512 bytes to the Dobbertin HD20 hard disc.

Label: HDSCHR or HDSCH (little faster).

ROM-number: B

Start address: &DF36 (HDSCHR), &DF3D (HDSCH)

Jump in conditions:
HDSCHR:
B = Number of head from 0 to 3.
C = Number of sector from 0 to 16 // &00 to &10.
DE = Number of cylinder from 0 to 613.
HL = Source address of 0.5 KB data.

HDSCH:
D = CCSSSSSS = 2 bit MSB cylinder, the lower 6 bit contain the sector number from 0 to 16.
E = LSB cylinder, lower 8 bit of the cylinder number from 0 to 613.
HL = Source address of 0.5 KB data.
XL = Number of head from 0 to 3.

Jump out conditions:
A = &00 => Success! Data has been written. HL = HL + &0200. Zero flag set
A = &02 => Sector was not found
A unequal to &00: An error has occurred
It the sector was written without an error the Z80 register A is set to &00 and register HL was increased by 0.5 KB (&0200). So HL points to byte after the end of the written data.

Manipulated: AF, BC, and HL. Using HDSCHR further: D and XL

Description: This OS function writes one sector of 512 bytes to the Dobbertin HD20 hard-disc. You can use two entry points: HDSCHR or HDSCH. The difference between them is in which Z80 registers the data has to be loaded before you call the OS function (look at Jump in conditions).
HDSCHR converts the provided data of the Z80 registers to another format and provides them for HDSCH. If you want to write all sectors to an track you should use HDSSPUR or HDSSP.
If you use the direct entry point HDSCH then you have to split the 10 bit cylinder number (0…613) into two registers. The lower eight bit must be provided in register E. The upper two bits must be written in the upper two bits of register D. But beware: Register D further contains in its lower six bits the sector number (from 0 to 16).

Attention: The hard disc must report ready. If it doesn't so and you try to use this OS function, this can lead to unknown errors.

READ THE DIRECTORY OF ONE HD20 PARTITION TO &4000

Short description: The DIRectory of one partition of the Dobbertin HD20 will be read to the RAM at address &4000.

Label: L_DIR_HD

ROM-number: B

Start address: &E156

Jump in conditions: A = Partition from 0 to 3.

Jump out conditions:
A = &00 => DIR was read correctly. DIR is located in RAM at &4000.
A = &02 => Sector not found. DIR sectors seem to contain an error.
A > &00 => Another error has occured.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL' and IX.
Further 1 KB is needed as buffer. The RAM between &B400 and &B7FF was manipulated.

Description: The Dobbertin HD-20 is divided into four partitions.

Every one of this four partitionen (FutureOS calls them I, J, K and L) has its own DIRectory. Every DIR has a length of 16 KB.

This OS function load the DIR of one partition (defined through the value 0, 1, 2 or 3 in the A register of the Z80) to the RAM address &4000. The RAM from &B800 to &BBFF is used as buffer.

If you switch in an 16 KB expansion RAM before you call this OS function, then the DIR will be loaded to that 16 KB E-RAM block between &4000 and &7FFF.

Attention: The buffer between &B800 and &BBFF was changed.

Write the DIRECTORY of one HD20 PARTITION beginning at &4000 to HD20

Short description: The DIRectory of one partition of the Dobbertin HD20 will be written back from RAM (start address &4000) to hard disc.

Label: S_DIR_HD

ROM-number: B

Start address: &E1E5

Jump in conditions: A = Partition 0...3 (corresponds I...L)
The DIR must be located at address &4000 in the actual RAM config.

Jump out conditions:
A = &00 => DIR was saved correctly.
A = &02 => Sector was not found.
A > &00 => Another error has occured.
The DIR was saved to hard disc if A contains &00.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL', IX.
Further 1 KB RAM (&B400-&B7FF) was used as buffer (content changed).

Description: This OS function is used to write the DIRectory of an HD20 hard-disc partition back to the Dobbertin hard-disc.

When calling the OS function the Z80 register A must contain the number of the partition: Valid values are 0, 1, 2 or 3, corresponding to the partitions I, J, K or L. Further the DIR must be located from &4000 to &7FFF (16 KB) in the actual switched on RAM.

After the return of this OS function the Z80 registe A contains a value, that reports about the success of the operation. Like usual the byte &00 tells that all went well, the DIR has been written back to the hard-disc HD20.

This OS function uses the fast write-track-OS function, so the RAM from &B400 to &B7FF was used as buffer.

Attention: If the Z80 register A is not equal to &00 after the return of the OS function, then an error has occured.

CONVERT A HD20 BLOCK-NUMBER TO CYLINDER, HEAD AND SECTOR

Short description: One block number of the Dobbertin HD20 hard disc will be converted into cylinder, head and sector.

Label: HDB2C

ROM-number: B

Start address: &E676

Jump in conditions:
HL = Number of block > 3 !!!
REG08_1 = Partition 0..3 (equals I, J, K and L).

Jump out conditions:
C = Sector &00 - &10 (0 - 17).
DE = Cylinder * 4 + head-number (lower 2 bits 1, 0).
HL = &0264 (= &0099 * 4).

Manipulated: AF, BC, DE and HL

Description: The DIRectorys of the Dobbertin HD20 hard disc contain 16 bit block-numbers. This OS function converts one logical block-number to a cylinder-number, head-number and sector-number.

When you call HDB2C the register HL must contain the logical block number. Further the RAM variable REG08_1 must contain the number of the partition from 0 to 3, which equals partition I, J, K or L.

After the OS function returns the register DE contains the cylinder number multiplicated by 4. That means that the bits 15-2 contain the cylinder number. Bits 1 and 0 of E contain the head number from 0 to 3. Further the Z80 register C contains the sector number between &00 and &10 (0 - 17 decimal).

Attention: The block number provided in HL MUST be bigger than 3. The first free block of the HD20 is in every partition number 4.
Why? Between Block 3 and 4 are 25.5 KB reserved as a kind of system track. That 25.5 KB quals 3 track of 8.5 KB each.

GENERATE A CYLINDER-HEAD-SECTOR TABLE (HD20)

Short description: Converts the block-numbers of a file to a cylinder-head-sector table.

Label: HD_TAB

ROM-number: B

Start address: &E4F1

Jump in conditions:
DE = First entry in the 32er DIR (&4000,&XX20,&XX40,&XX60 .. &7FE0)
HL = Target address for the Cylinder-Head-Sector table
REG08_1 = Partition 0..3 (=I..L)

Jump out conditions: Table was created.

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

Description: If you load a file or save a file under FutureOS, then the OS works with Load- or Save- tables also called Cylinder-Head-Sector tables. This technique leads to strong speedup. Before the OS can use a file it converts all its block numbers to Cylinder, Head and Sectors. This data is then collected in such a Cylinder, Head, Sector table.

Construction of a Load/Save Cylinder, Head, Sector table for the HD20:

1. 16 bit:
- Cylinder-number * 4 + head(0..3) or ...
- &FFFF to mark the end of the table.

2. 8 bit:
- Number of different sectors (1..16) or ...
- &FF to mark the whole track, use all 17 sectors of the track.

3. X * 8 bit: (if the "Number of sectors" byte is <> &FF)
- sector-numbers, with ascending numbers, up to 16 * 8 bit.

4. Now the next "Cylinder * 4 + head" number follows (look 1.) or &FFFF marks the end of the table.

When calling HD_TAB the register DE must point to the first entry of a file. It must be possible to divide this address by 32. It can have to following values: &XX00, &XX20, &XX40, &XX60, &XX80, &XXA0, &XXC0 or &XXE0.
The target address of the table (provided in HL) can be choose freely.
But you should think that this table can get very long when using very long files.
Further the RAM variable REG08_1 must contain the partition number from 0 to 3.
After calling HD_TAB the table will be generated.

Attention: The source address must point to a valid entry in the DIR of the hard disc. Bytes after the target address will be overwritten.

LOAD A FILE FROM HARD DISC HD20 (max. 40 KB)

Short description: Loads a file of up to 40 KB from HD20 hard disc.

Label: LHD4

ROM-number: B

Start address: &E254

Jump in conditions:
HL = Address of the Cylinder-Head-Sector table of the file.
REG16_1 = Target address in RAM.

Jump out conditions:
A = &00 => Success! File was loaded correctly.
A = &02 => Error! Sector was not found.
A unequal &00 => Error has occured.
File has been read to RAM if register A=&00.

Manipulated: AF,BC,DE,HL,IX,IY and RAM variables REG16_0, REG16_1.

Description: The OS function LHD4 load a file of up to 40 KB from the Dobbertin HD20 hard disc to RAM. If you don't need the area of the file-tagging-bytes, then you can load up to 44 KB (&0000-&AFFF).

Before you call the OS function you have to load register HL with the start address of the previously generated Cylinder-Head-Sector table.

The target address of the file must be provided in the RAM variable REG16_1.

After calling the OS function and loading the file, register A reports eventual errors. If A is zero no error has occured.

Attention: Be carefull that the file doesn't overwrite RAM beyond &A000(FTBs) or &B000(system RAM!!!). The hard disc HD20 must report "Ready" status before using this OS function.

LOAD A FILE FROM HARD DISC HD20 (max. 512 KB) TO E-RAM

Short description: Loads a file of up to 512 KB from HD20 hard disc to expansion RAM.

Label: LHD6

ROM-number: B

Start address: &E355

Jump in conditions:
HL = Address of the Cylinder-Head-Sector table of the file.
REG16_1 = Target address in expansion RAM (&4000-&7F00).
AKT_RAM = Target expansion RAM block, where file should be loaded.

Jump out conditions:
A = &00 => Success! File was loaded correctly.
A = &02 => Error! Sector was not found.
A unequal &00 => Error has occured.
File has been read to RAM if register A=&00.

Manipulated: AF, BC, DE, HL, IX, IY, the RAM variables REG16_0, REG16_1, AKT_RAM and the RAM configuration. Further the RAM from &8000 to &9FFF (8 KB track buffer).

Description: The OS function LHD6 allows to load a file of up to 512 KB to the expansion RAM of the CPC. The Dobbertin hard disc HD20 is used.

Before you call LHD6 you have to load register HL with the start address of a previously generated Cylinder-Head-Sector table. The target address (where the file should be loaded to expansion RAM) must be provided in the RAM variable REG16_1. Further the physical number of the 16 KB block of the expansion RAM must be written in RAM variable AKT_RAM. Like usual the RAM can be selected through the values &C4, &C5, &C6, &C7, &CC, &CD, &CE, &CF, &D4 ... &FF.

If the file has been loaded correctly to the expansion RAM of the CPC the register A will returen the value zero. Else an error has occured.

Attention: The target expansion RAM (to which a file shall be loaded) must exist physicalls. The hard disc HD20 must report "Ready" status before using this OS function.

SAVE A FILE TO HD20 HARD DISC (max. 64 KB)

Short description: Saves a file of up to 64 KB to HD20 hard disc.

Label: SHD4

ROM-number: B

Start address: &E3CB

Jump in conditions:
HL = Address of the Cylinder-Head-Sector table of the file.
REG16_1 = Source address of data in RAM.

Jump out conditions:
A = &00 => Success! File was saved correctly.
A = &02 => Error! Sector was not found.
A unequal &00 => Error has occured.
File has been saved to hard disc HD20 if register A=&00.

Manipulated: AF, BC, DE, HL, IX and RAM variables REG16_0, REG16_1.

Description: SHD4 saves a file of up to 64 KB from RAM to the HD20 hard disc (Dobbertin). The actual RAM configuration is used.

Register HL must contain the start address of the previously generated Cylinder-Head-Sector table. The source address of the data, that shall be written to the hard disc, must be provided in RAM variable REG16_1.

Now you can call the OS function SHD4. After the OS function returns the A register reports if an error has ocured. If A is zero, then all went well.

Attention: The hard disc HD20 must report "Ready" status before using this OS function.

SAVE A FILE OF UP TO 512 KB FROM E-RAM TO HD20 HARD DISC

Short description: Saves a file of up to 512 KB to HD20 hard disc.

Label: SHD6

ROM-number: B

Start address: &E423

Jump in conditions:
HL = Address of the Cylinder-Head-Sector table of the file.
REG16_1 = Source address of data in expansion RAM.
AKT_RAM = Source 16 KB RAM block, first block to be saved from.

Jump out conditions:
A = &00 => Success! File was saved correctly.
A = &02 => Error! Sector was not found.
A unequal &00 => Error has occured.
File has been saved to hard disc HD20 if register A=&00.

Manipulated: AF, BC, DE, HL, BC', DE', HL', IX, the variables REG16_0, REG16_1, AKT_RAM and the RAM configuration. Further the RAM area between &8000 and &9FFF (8 KB track buffer).

Description: SHD6 saves a file of up to 512 KB from the expansion RAM to the Dobbbertin HD20 hard disc.

Register HL must contain the start address of the previously generated Cylinder-Head-Sector table. The source address of the data, that shall be written to the hard disc, must be provided in RAM variable REG16_1.

Further the physical number of the first 16 KB block of the expansion RAM, from which shall be saved, must be written in RAM variable AKT_RAM. Like usual the RAM can be selected through the values &C4, &C5, &C6, &C7, &CC, &CD, &CE, &CF, &D4 ... &FF.

Now you can call the OS function SHD6. After the OS function returns the A register reports if an error has ocured. If A is zero, then all went well and the file has been saved correctly.

Attention: The hard disc HD20 must report "Ready" status before using this OS function.

SELECT NEXT 16 KB EXPANSION RAM BLOCK (in AKT_RAM)

Short description: Select the next 16 KB expansion RAM block. Variable AKT_RAM will hold the new RAM configuration. Up to 8 MB RAM.

Label: NXX_ERM

ROM-number: B

Start address: &E72B

Jump in conditions: RAM variable AKT_RAM must hold a correct value.

Jump out conditions: Variable AKT_RAM contains the physical RAM select of the next upper 16 KB expansion RAM block, which is switched in between &4000 and &7FFF.

Manipulated: F, BC and RAM variable AKT_RAM.

Description: OS function NXX_ERM is used to manage up to 8 MB expansion RAM. Todays biggest RAM expansion contains up to 4 MB.
The OS function reads the actual RAM configuration from the content of RAM variable AKT_RAM, it selects the next upper 16 KB expansion RAM bock, banks that block in between &4000 and &7FFF and saves the new RAM configuration to system variable AKT_RAM.

The single 16 KB blocks of the 4 MB expansion RAM are physically (RAM configuration) numbered in ascending order like shown here:

```
&7FC4, &7FC5, &7FC6, &7FC7, &7FCC, &7FCD, &7FCE, &7FCF,
&7FD4, &7FD5, &7FD6, &7FD7, &7FDC, &7FDD, &7FDE, &7FDF,
&7FE4, &7FE5, &7FE6, &7FE7, &7FEC, &7FED, &7FEE, &7FEF,
&7FF4, &7FF5, &7FF6, &7FF7, &7FFC, &7FFD, &7FFE, &7FFF,

&7EC4, &7EC5, &7EC6, &7EC7, &7ECC, &7ECD, &7ECE, &7ECF,
&7ED4, &7ED5, &7ED6, &7ED7, &7EDC, &7EDD, &7EDE, &7EDF,
&7EE4, &7EE5, &7EE6, &7EE7, &7EEC, &7EED, &7EEE, &7EEF,
&7EF4, &7EF5, &7EF6, &7EF7, &7EFC, &7EFD, &7EFE, &7EFF,

&7DC4, &7DC5, &7DC6, &7DC7, &7DCC, &7DCD, &7DCE, &7DCF,
&7DD4, &7DD5, &7DD6, &7DD7, &7DDC, &7DDD, &7DDE, &7DDF,
&7DE4, &7DE5, &7DE6, &7DE7, &7DEC, &7DED, &7DEE, &7DEF,
&7DF4, &7DF5, &7DF6, &7DF7, &7DFC, &7DFD, &7DFE, &7DFF,

&7CC4, &7CC5, &7CC6, &7CC7, &7CCC, &7CCD, &7CCE, &7CCF,
&7CD4, &7CD5, &7CD6, &7CD7, &7CDC, &7CDD, &7CDE, &7CDF,
&7CE4, &7CE5, &7CE6, &7CE7, &7CEC, &7CED, &7CEE, &7CEF,
&7CF4, &7CF5, &7CF6, &7CF7, &7CFC, &7CFD, &7CFE, &7CFF,

... &7BXX ... &7AXX ... &79XX ...

&78C4, &78C5, &78C6, &78C7, &78CC, &78CD, &78CE, &78CF,
&78D4, &78D5, &78D6, &78D7, &78DC, &78DD, &78DE, &78DF,
&78E4, &78E5, &78E6, &78E7, &78EC, &78ED, &78EE, &78EF,
&78F4, &78F5, &78F6, &78F7, &78FC, &78FD, &78FE, &78FF.
```

Attention: Please use RAM beyond the 4 MB only if it is physically connected to your CPC.

SELECT NEXT 16 KB EXPANSION RAM BLOCK (REGISTER BC)

Short description: Select the next 16 KB expansion RAM block. Register BC will hold the new RAM configuration. Up to 8 MB RAM.

Label: NXT_ERM

ROM-number: B

Start address: &FE81

Jump in conditions: Register BC must hold the RAM configuration.

Jump out conditions: Register BC contains the physical RAM select of the next upper 16 KB expansion RAM block, which is switched in between &4000 and &7FFF.

Manipulated: F and BC.

Description: OS function NXT_ERM is used to manage up to 8 MB expansion RAM. Todays biggest RAM expansion contains up to 4 MB.
The OS function reads the actual RAM configuration from register BC, it selects the next upper 16 KB expansion RAM bock, banks that block in between &4000 and &7FFF and saves the new RAM configuration to register BC.

The single 16 KB blocks of the 4 MB expansion RAM are physically (RAM configuration) numbered in ascending order like shown here:

```
&7FC4, &7FC5, &7FC6, &7FC7, &7FCC, &7FCD, &7FCE, &7FCF,
&7FD4, &7FD5, &7FD6, &7FD7, &7FDC, &7FDD, &7FDE, &7FDF,
&7FE4, &7FE5, &7FE6, &7FE7, &7FEC, &7FED, &7FEE, &7FEF,
&7FF4, &7FF5, &7FF6, &7FF7, &7FFC, &7FFD, &7FFE, &7FFF,
&7EC4, &7EC5, &7EC6, &7EC7, &7ECC, &7ECD, &7ECE, &7ECF,
&7ED4, &7ED5, &7ED6, &7ED7, &7EDC, &7EDD, &7EDE, &7EDF,
&7EE4, &7EE5, &7EE6, &7EE7, &7EEC, &7EED, &7EEE, &7EEF,
&7EF4, &7EF5, &7EF6, &7EF7, &7EFC, &7EFD, &7EFE, &7EFF,
&7DC4, &7DC5, &7DC6, &7DC7, &7DCC, &7DCD, &7DCE, &7DCF,
&7DD4, &7DD5, &7DD6, &7DD7, &7DDC, &7DDD, &7DDE, &7DDF,
&7DE4, &7DE5, &7DE6, &7DE7, &7DEC, &7DED, &7DEE, &7DEF,
&7DF4, &7DF5, &7DF6, &7DF7, &7DFC, &7DFD, &7DFE, &7DFF,
&7CC4, &7CC5, &7CC6, &7CC7, &7CCC, &7CCD, &7CCE, &7CCF,
&7CD4, &7CD5, &7CD6, &7CD7, &7CDC, &7CDD, &7CDE, &7CDF,
&7CE4, &7CE5, &7CE6, &7CE7, &7CEC, &7CED, &7CEE, &7CEF,
&7CF4, &7CF5, &7CF6, &7CF7, &7CFC, &7CFD, &7CFE, &7CFF,
... &7BXX ... &7AXX ... &79XX ...
&78C4, &78C5, &78C6, &78C7, &78CC, &78CD, &78CE, &78CF,
&78D4, &78D5, &78D6, &78D7, &78DC, &78DD, &78DE, &78DF,
&78E4, &78E5, &78E6, &78E7, &78EC, &78ED, &78EE, &78EF,
&78F4, &78F5, &78F6, &78F7, &78FC, &78FD, &78FE, &78FF.
```

Attention: Please use RAM beyond the 4 MB only if it is physically connected to your CPC.

SELECT PREVIOUS 16 KB EXPANSION RAM BLOCK (IN AKT_RAM)

Short description: Select the previous 16 KB expansion RAM block. The variable AKT_RAM will hold the new RAM configuration. Up to 8 MB RAM.

Label: LXX_ERM

ROM-number: B

Start address: &E745

Jump in conditions: RAM variable AKT_RAM must hold a correct value.

Jump out conditions: The Sign flag reports about the success:
Sign flag = M, Error, lower end of RAM was reached.
Sign flag = P, all went well, new RAM block selected. In this case:
Variable AKT_RAM contains the physical RAM select of the previous 16 KB expansion RAM block, which is switched in between &4000 and &7FFF.

Manipulated: F, BC and RAM variable AKT_RAM.

Description: OS function LXX_ERM is used to manage up to 8 MB expansion RAM. Todays biggest RAM expansion contains up to 4 MB.
The OS function reads the actual RAM configuration from the content of RAM variable AKT_RAM, it selects the previous 16 KB expansion RAM bock, banks that block in between &4000 and &7FFF and saves the new RAM configuration to system variable AKT_RAM.

The single 16 KB blocks of the 4 MB expansion RAM are physically (RAM configuration) numbered in ascending order like shown here:

```
&7FC4, &7FC5, &7FC6, &7FC7, &7FCC, &7FCD, &7FCE, &7FCF,
&7FD4, &7FD5, &7FD6, &7FD7, &7FDC, &7FDD, &7FDE, &7FDF,
&7FE4, &7FE5, &7FE6, &7FE7, &7FEC, &7FED, &7FEE, &7FEF,
&7FF4, &7FF5, &7FF6, &7FF7, &7FFC, &7FFD, &7FFE, &7FFF,
&7EC4, &7EC5, &7EC6, &7EC7, &7ECC, &7ECD, &7ECE, &7ECF,
&7ED4, &7ED5, &7ED6, &7ED7, &7EDC, &7EDD, &7EDE, &7EDF,
&7EE4, &7EE5, &7EE6, &7EE7, &7EEC, &7EED, &7EEE, &7EEF,
&7EF4, &7EF5, &7EF6, &7EF7, &7EFC, &7EFD, &7EFE, &7EFF,
&7DC4, &7DC5, &7DC6, &7DC7, &7DCC, &7DCD, &7DCE, &7DCF,
&7DD4, &7DD5, &7DD6, &7DD7, &7DDC, &7DDD, &7DDE, &7DDF,
&7DE4, &7DE5, &7DE6, &7DE7, &7DEC, &7DED, &7DEE, &7DEF,
&7DF4, &7DF5, &7DF6, &7DF7, &7DFC, &7DFD, &7DFE, &7DFF,
&7CC4, &7CC5, &7CC6, &7CC7, &7CCC, &7CCD, &7CCE, &7CCF,
&7CD4, &7CD5, &7CD6, &7CD7, &7CDC, &7CDD, &7CDE, &7CDF,
&7CE4, &7CE5, &7CE6, &7CE7, &7CEC, &7CED, &7CEE, &7CEF,
&7CF4, &7CF5, &7CF6, &7CF7, &7CFC, &7CFD, &7CFE, &7CFF,
... &7BXX ... &7AXX ... &79XX ...
&78C4, &78C5, &78C6, &78C7, &78CC, &78CD, &78CE, &78CF,
&78D4, &78D5, &78D6, &78D7, &78DC, &78DD, &78DE, &78DF,
&78E4, &78E5, &78E6, &78E7, &78EC, &78ED, &78EE, &78EF,
&78F4, &78F5, &78F6, &78F7, &78FC, &78FD, &78FE, &78FF.
```

Attention: Please use RAM beyond the 4 MB only if it is physically connected to your CPC.

SELECT PREVIOUS 16 KB EXPANSION RAM BLOCK (REGISTER BC)

Short description: Select the previous 16 KB expansion RAM block. The register BC will hold the new RAM configuration. Up to 8 MB RAM.

Label: LST_ERM

ROM-number: B

Start address: &FEA3

Jump in conditions: Register BC must hold the RAM configuration.

Jump out conditions: The Sign flag reports about the success:
Sign flag = M, Error, lower end of RAM was reached.
Sign flag = P, all went well, new RAM block selected. In this case:
Register BC contains the physical RAM select of the previous 16 KB expansion RAM block, which is switched in between &4000 and &7FFF.

Manipulated: F and BC.

Description: OS function LST_ERM is used to manage up to 8 MB expansion RAM. Todays biggest RAM expansion contains up to 4 MB.
The OS function reads the actual RAM configuration from register BC, it selects the previous 16 KB expansion RAM bock, banks that block in (&4000-&7FFF) and saves the new RAM configuration to register BC.

The single 16 KB blocks of the 4 MB expansion RAM are physically (RAM configuration) numbered in ascending order like shown here:

```
&7FC4, &7FC5, &7FC6, &7FC7, &7FCC, &7FCD, &7FCE, &7FCF,
&7FD4, &7FD5, &7FD6, &7FD7, &7FDC, &7FDD, &7FDE, &7FDF,
&7FE4, &7FE5, &7FE6, &7FE7, &7FEC, &7FED, &7FEE, &7FEF,
&7FF4, &7FF5, &7FF6, &7FF7, &7FFC, &7FFD, &7FFE, &7FFF,

&7EC4, &7EC5, &7EC6, &7EC7, &7ECC, &7ECD, &7ECE, &7ECF,
&7ED4, &7ED5, &7ED6, &7ED7, &7EDC, &7EDD, &7EDE, &7EDF,
&7EE4, &7EE5, &7EE6, &7EE7, &7EEC, &7EED, &7EEE, &7EEF,
&7EF4, &7EF5, &7EF6, &7EF7, &7EFC, &7EFD, &7EFE, &7EFF,

&7DC4, &7DC5, &7DC6, &7DC7, &7DCC, &7DCD, &7DCE, &7DCF,
&7DD4, &7DD5, &7DD6, &7DD7, &7DDC, &7DDD, &7DDE, &7DDF,
&7DE4, &7DE5, &7DE6, &7DE7, &7DEC, &7DED, &7DEE, &7DEF,
&7DF4, &7DF5, &7DF6, &7DF7, &7DFC, &7DFD, &7DFE, &7DFF,

&7CC4, &7CC5, &7CC6, &7CC7, &7CCC, &7CCD, &7CCE, &7CCF,
&7CD4, &7CD5, &7CD6, &7CD7, &7CDC, &7CDD, &7CDE, &7CDF,
&7CE4, &7CE5, &7CE6, &7CE7, &7CEC, &7CED, &7CEE, &7CEF,
&7CF4, &7CF5, &7CF6, &7CF7, &7CFC, &7CFD, &7CFE, &7CFF,

... &7BXX ... &7AXX ... &79XX ...

&78C4, &78C5, &78C6, &78C7, &78CC, &78CD, &78CE, &78CF,
&78D4, &78D5, &78D6, &78D7, &78DC, &78DD, &78DE, &78DF,
&78E4, &78E5, &78E6, &78E7, &78EC, &78ED, &78EE, &78EF,
&78F4, &78F5, &78F6, &78F7, &78FC, &78FD, &78FE, &78FF.
```

Attention: Please use RAM beyond the 4 MB only if it is physically connected to your CPC.

READ 128 BYTES FROM A I/O PORT

Short description: This OS function reads blocks of 128 bytes from an I/O port.

Label: F_INP

ROM-number: B

Start address: &DD84

Jump in conditions:
 A = Number of 128 byte block which shall be read
BC = 16 bit port address from which data shall be read
HL = Target address of data to be read

Jump out conditions: Data was read

Manipulated: AF, HL and the memory area after HL

Description: This function allows to read 1-256 blocks of 128 bytes each from an 16 bit I/O address to RAM.

Attention: A maximum of 32 KB can be read at once.

WRITE A BLOCK OF 128 BYTES TO AN I/O PORT

Short description: This OS function sends blocks of 128 bytes to an 16 bit I/O port.

Label: F_OUT

ROM-number: B

Start address: &DFC2

Jump in conditions:
A = Number of 128 byte blocks to be send to the I/O address
BC = 16 bit I/O address to which the data shall be sent
HL = Source address at which the data starts

Jump out conditions: Data was sent

Manipulated: AF, HL

Description: This OS function allows to send 1-256 blocks of 128 bytes each to a 16 bit I/O port address.

Attention: A maximum of 32 KB can be sent at once

The newest version of this file (API-B-EN.DOC) can be ordered from FutureSoft@gmx.de or download it at FutureSoft's home page.

FutureOS home page: http://www.FutureOS.de