Datei: ‘API-C-DE.DOK‘ (60 Seiten) Stand: 18. September, Jahr des CPC 2023

Datei zur Dokumentation aller Future Operating System Funktionen des aktuellen OS System .8 Die für den Anwender oder Programmierer wichtigen OS Funktionen aus ROM C werden erklärt und lokalisiert.

Alle OS Funktionen werden in folgender Form beschrieben:

1. Kurzbeschreibung: In einem Satz wird kurz die Funktion der OS Funktion beschrieben.

2. Label: Mit diesem Label wird die OS Funktion im Source Code bezeichnet. In der mitgelieferten Label-Bibliothek (siehe Datei ‚**#EQU-API.DEU**‘) mit den jeweils aktuellen ROM Adressen findet ebenfalls dieses Label Verwendung. Aus Gründen der Kompatibilität sollte man in eigenen Programmen alle OS- / System-Funktionen (und System-Variablen) stets mit diesen Labels ansprechen.

3. ROM-Nummer: Hier ist die logische ROM - Nummer des FutureOS ROMs angegeben, in dem die entsprechende OS Funktion zu finden ist.
Manche OS Funktionen kommen, wegen ihrer Kürze, in mehreren ROMs vor, dann sind hier auch mehrere ROM Nummern angegeben.
Da die ROM Nummer logisch zu verstehen ist, sollte man sie nicht mit dem physikalischen ROM Select gleichsetzen. Wie man die physikalische ROM Nummer eines der FutureOS ROMs ermittelt wird im Handbuch erklärt.

4. Startadresse: Gibt die Einsprung-Adresse der OS Funktion an. Falls diese in mehreren ROMs vorkommt, wird hier auch eine entsprechende Anzahl von Adressen angegeben.

5. Einsprungbedingungen: Die Einsprungbedingungen der OS Funktion werden beschrieben und es wird sowohl auf Registerinhalte als auch auf RAM-Variablen eingegangen.

6. Aussprungbedingungen: Die Aussprungbedingungen der OS-Funktion werden beschrieben und es wird sowohl auf Registerinhalte als auch auf RAM-Variablen eingegangen.

7. Manipuliert: Es werden alle manipulierten oder zerstörten Register und RAM-Variablen wiedergegeben. Manchmal werden auch manipulierte Pheripheriebausteine wiedergegeben.

8. Beschreibung: Es folgt eine vollständige Erklärung der Anwendung und Wirkungsweise der beschriebenen OS Funktion.

9. Bitte Beachten: Es werden einige wichtige Details kurz erklärt.
Dies ist besonders wichtig, da alle OS Funktionen kompromißlos auf Höchstgeschwindigkeit getrimmt worden sind. Bei falscher Handhabung kann es zu Problemen mit der Systemstabilität kommen.

Das ROM C stellt diverse OS Funktionen zur Verfügung, z.B. zum Bewegen von Daten, für die DIR / IHV Verwaltung, Laden und Speichern usw. Viele High-Level OS Funktionen befinden sich in diesem ROM.

Die jeweils aktuelle Version dieser Datei (‚API-C-DE.DOK') ist auch im Internet erhältlich:

http://www.FutureOS.de

EXTREM SCHNELLES FÜLLEN EINES SPEICHERBEREICHS

Kurzbeschreibung: Diese OS Funktion stellt die schnellste Art dar einen größeren Speicherbereich mit einem Byte oder einem zwei Byte Wort zu füllen.

Label: F_FILL8 (Speicher mit einem Byte füllen, 8 Bit) oder F_FILL6 (Speicher mit einem zwei Byte Wort füllen, 16 Bit)

ROM-Nummer: C

Startadresse: &C01E (F_FILL8) oder &C01F (F_FILL6)

Einsprungbedingungen: BC = Länge des zu füllenden Speicherbereichs
HL = Startadresse des zu füllenden Speicherbereichs.
D = Wert mit dem gefüllt werden soll, bei der 8 Bit Version F_FILL8.
..... bzw. .....
DE = Wert mit dem gefüllt werden soll, bei der 16 Bit Version F_FILL6.

Aussprungbedingungen: Der Speicherbereich ist gefüllt.

Manipuliert: AF, B, HL, HL' und bei F_FILL8 zusätzlich E

Beschreibung: Diese OS Funktion füllt einen Speicherbereich in kürzester Zeit. Um so größer der zu füllende Speicherbereich ist, desto schneller wird die OS Funktion. Pro Byte vergehen min. 1.5 µs.

Man verwendet die OS Funktion z.B. um die 16 KB des Bildschirmspeichersmit &00 zu füllen, sprich den Bildschirm zu löchen.

Vor Aufruf der OS Funktion läd man HL mit der Adresse des ersten zu füllenden Bytes, BC erhält die Anzahl der zu füllenden Bytes. Die OS Funktion füllt den Speicher aufsteigend. Es wird also der Speicher zwischen HL und HL + BC gefüllt. Dies geschieht entweder mit einem 8 Bit Wert, hierzu verwendet man das Label F_FILL8, oder man füllt den Speicher mit einem 16 Bit Wert, dies geschieht dann mit F_FILL6. Dabei wird das Low- immer vor dem High-Byte in den Speicher geschrieben.

Will man mit 8 Bit füllen so läd man vor Aufruf der OS Funktion das D Register mit dem entsprechendem 8 - Bit Wert. Bei Verwendung eines 16 Bit Wertes läd man diesen in das DE Register. Beispiele:

```
TEST_FILL_8    LD BC,&4000     ;&4000 Bytes werden

               LD   D,&88      ;mit dem Byte &88 gefüllt.
               LD   HL,&2000   ;Füllen ab Adresse &2000
               CALL F_FILL8
```

```
TEST_FILL_16   LD   BC,&4000 ;&4000 Bytes werden

               LD   DE,&1234 ;mit dem Doppel-Byte &1234 gefüllt
               LD   HL,&2000 ;Füllen ab Adresse &2000
               CALL F_FILL6
```

Falls ROM C vor Aufruf der OS Funktion nicht eingeblendet ist, so sind folgende Befehle als erste Zeile einzufügen:

```
LD  BC,(&FF0D)      ;physikalische Nummer von ROM C holen
OUT (C),C           ;und ROM C einblenden!
```

Bitte Beachten: Diese OS Funktion überschreibt jeden ihr angegebenen Speicherbereich. Wenn man also ein Register vor dem Aufruf versehentlich mit einer falschen Adresse läd, dann kann dies die Stabilität des Systems gefährden.
Beim 16-Bit füllen wird zuerst das Low-Bytes (Register E) und dann das High-Byte (Register D) geschrieben.

KOPIEREN EINES SPEICHERBEREICHES GRÖßER 256 BYTES

Kurzbeschreibung: Diese OS Funktion ermöglicht es einen Speicherbereich von mindestens 256 Bytes extrem schnell zu kopieren.

Label: F_MOVE

ROM-Nummer: C

Startadresse: &C0C8

Einsprungbedingungen: BC = Länge des zu kopierenden Speicherblocks. Diese Länge muß mindestens &100 betragen. Das B Register darf nicht gleich Null sein.
DE = Zieladresse des zu kopierenden Blocks
HL = Quelladresse des zu kopierenden Blocks

Aussprungbedingungen: Der Block wurde kopiert.

Manipuliert: AF, BC, DE und HL

Beschreibung: Ein Speicherbereich der mindestens 256 Bytes enthält wird mit Höchstgeschwindigkeit kopiert. Dabei kann im Speicher auf und abwärts kopiert werden. Quell- und Zielblock können sich dabei überschneiden. Vor Aufruf der OS Funktion muß HL auf des erste Byte des Quellblocks zeigen, DE muß auf das erste Bytes des Zielblocks zeigen. Das Register BC enthält die Anzahl der zu kopierenden Bytes es muß aber mit mindestens 256 = &100 gefüllt sein. Kleinere Blöcke bitte mit LDDR oder mit LDIR kopieren.

Beispiel:

```
HOLE_BILD LD   DE,&C000  ;Zieladresse (zu kopierender Block)

          LD   HL,&4000  ;Quelladresse (zu kopierender Block)
          LD   B,H       ;Länge des zu kopierenden Blocks
          LD   C,L       ;entspricht zufällig HL, beide &4000
          CALL F_MOVE    ;schnellstens kopieren!
```

Bitte Beachten: Ist bei Aufruf der OS Funktion das Register BC mit einem Wert kleiner als 256 = &100 geladen, so werden knapp 64K kopiert und das System wird wohl abschmieren.

BLOCK SCHNELL SPEICHERAUFWÄRTS KOPIEREN

Kurzbeschreibung: Diese OS Funktion ist ein Teil von F_MOVE, sie dient dazu Blöcke von mindestens 256 Bytes im Speicher nach oben zu kopieren.

Label: LDD_256

ROM-Nummer: C

Startadresse: &C2E8

Einsprungbedingungen:
BC = Blocklänge in Bytes, mindestens 256 = &100.
DE > HL
DE = oberes Ende des Zielblocks.
HL = oberes Ende des Quellblocks.

Aussprungbedingungen: Der Block ist kopiert worden.

Manipuliert: AF, BC, DE und HL

Beschreibung: Dieser OS Funktion dient dazu einen Block mit der Mindestlänge von 256 = &100 Bytes zu übertragen. Dabei müß der Zeilblock oberhalb des Quellblocks liegen. Wenn der Block nicht teilweise auf sich selbst kopiert wird, dann kann der Quellblock an eine beliebige Adresse kopierte werden, nur eben nicht auf sich selber. Will man den Block also teilweise auf sich selber kopieren, dann immer nur speicheraufwärts.

Bitte Beachten: Dei Blocklänge BC muß mindestens 256 = &100 Bytes sein, da daß System sonst mit hoher Warscheinlichkeit abstürtzen würde.

BLOCK SCHNELL SPEICHERABWÄRTS KOPIEREN

Kurzbeschreibung: Diese OS Funktion ist ein Teil von F_MOVE, sie dient dazu um Speicherbereiche von minimal 256 = &100 Bytes speicherabwärts zu kopieren.

Label: LDI_256

ROM-Nummer: C

Startadresse: &C0D0

Einsprungbedingungen:
BC = Blocklänge in Bytes, mindestens 256 = &100.
DE < HL
DE = unters Ende des Zielblocks.
HL = unters Ende des Quellblocks.

Aussprungbedingungen: Der Block ist kopiert worden.

Manipuliert: AF, BC, DE und HL

Beschreibung: Dieser OS Funktion dient dazu einen Block mit der Mindestlänge von 256 = &100 Bytes zu übertragen. Dabei müß der Zeilblock unterhalb des Quellblocks liegen. Wenn der Block nicht teilweise auf sich selbst kopiert wird, dann kann der Quellblock an eine beliebige Adresse kopierte werden, nur eben nicht auf sich selber. Will man den Block also teilweise auf sich selber kopieren, dann immer nur speicherabwärts.

Bitte Beachten: Die Blocklänge in BC muß mindestens 256 = &100 Bytes lang sein, ist dies nicht der Fall so wird das System mit an Sicherheit grenzender Warscheinlichkeit abstürtzen.

ZEICHENSATZ VOM ROM INS RAM KOPIEREN

Kurzbeschreibung: OS Funktion kopiert den ROM-Zeichensatz ins RAM.

Label: INRZ

ROM-Nummer: C

Startadresse: &C9AD

Einsprungbedingungen: RAMCHAR bestimmt den Bildschirm-Modus.

Aussprungbedingungen: Zeichensatz von ROM ins RAM kopiert.

Manipuliert: AF, BC, DE, HL, RAM &3800-&3FFF, unteres RAM aktiv

Beschreibung: Diese OS Funktion kopiert den ROM Zeichensatz (alle 256 Zeichen) von &3800 bis &3FFF and die selben Adressen ins untere RAM. Nach der Rückkehr ist das untere RAM eingeblendet. INRZ kann also auch vom unterem RAM aus aufgerufen werden.

Der Video-Modus wird entsprechend der OS Variable RAMCHAR gesetzt.

Bitte Beachten: Der RAM Speicherbereich zwischen &3800 und &3FFF wurde überschrieben.

SORTIEREN EINES ZUVOR EINGELESENEN DIRECTORYS

Kurzbeschreibung: Ein zuvor eingelesenes Inhaltsverzeichnis einer Diskette wird sortiert.

Label: TS_D_IN

ROM-Nummer: C

Startadresse: &FDA7

Einsprungbedingungen: D = aufgerundetes Highbyte des oberen Endes des Speicherbereichs in dem sich das zu sortierende Inhaltsverzeichnis befindet.

REG16_1 = Länge des Inhaltsverzeichnisses in Bytes. z.B. &800 = 2048 Bytes beim Data Format oder &1000 = 4096 Bytes für das Vortex Format.
Es spielt keine Rolle zu wieviel Prozent das Directory gefüllt ist, die absolute Länge ist entscheident.

REG16_2 = Zeiger auf den Anfang des zu sortierenden Inhaltsverzeichnisses. Es ist das erste, unterste Byte gemeint.

REG16_3 = REG16_2. Beide Variablen müßen den gleichen Wert enthalten.

REG16_5 = Zeiger auf den Anfang eines Pufferspeicherbereichs, der genau so groß ist wie das zu sortierende Inhaltsverzeichnis.

REG16_6 = REG16_5. Beide Variablen müßen den gleichen Wert enthalten.

REG16_7 = Zeiger auf das Ende des durch REG16_5 adressierten Pufferbereichs. Der Programmierer kann diesen Wert durch die Addition von REG16_5 + REG16_1 erhalten, muß ihn aber selber in diese 16 Bit RAM Variable laden.

Aussprungbedingungen: Das Inhaltsverzeichnis ist sortiert.

Manipuliert: AF, BC, DE, HL, REG16_0, REG16_3, REG16_4 und REG16_6

Beschreibung: Diese OS Funktion führt der Reihe nach folgende Aktionen aus. Zuerst werden alle leeren Inhaltsverzeichnis-Einträge ausgefiltert, dann werden die gültigen Einträge numeroalphabetisch sortiert. Zuletzt wird der restliche Platz im Inhaltsverzeichnis mit dem Wert &E5 aufgefüllt.

Am Beispiel des Directorys einer Data-formatierten Disk wird hier die Wirkungsweise besprochen. Angenommen das Inhaltsverzeichnis wurde von &6000 bis &6800 eingelesen. Es ist also &0800 Bytes lang. Deshalb schreiben wir in REG16_1 (Länge) den Wert &0800.

In REG16_2 und REG16_3 wird der Wert &6000 geladen, das ist der Start des DIRs. Desweiteren benötigt die OS Funktion zum sortieren einen Pufferbereich in der selben Länge des Directoryblocks. Das das DIR also &0800 Bytes lang ist brauchen wir einen &0800 Bytes lange Pufferbereich, wir verwenden z.B. den Bereich von &7000 bis &7800.

In REG16_5 und REG16_6 laden wir also den Wert &7000 (Start Pufferbereich).

Und in REG16_7 laden wir nun den Wert &7800.

Nun müssen wir noch das Prozessor-Register D mit dem Highbyte des unsortierten Inhaltsverzeichnisses laden. Dieser Wert erhalten wir wenn wir die Startadr. des DIR und die Länge addieren, und anschließend durch 256 = &0100 teilen. Also &6000 + &0800 = &6800. Und &6800/&0100 = &68. Register D wird mit &68 geladen.

So und nun können wir diese SortierOS Funktion auch schon aufrufen.

Nachdem diese OS Funktion das Inhaltsverzeichnis sortiert hat steht das neue, sortierte Inhaltsverzeichnis dort wo auch das alte stand.

Bitte Beachten: Es muß ein Pufferbereich zur Verfügung stehen, der genauso groß ist, wie das zu sortierende Inhaltsverzeichnis. Alle Adressen sind so zu wählen, daß sie durch 256 zu teilen sind, also z.B. &100, &200, &300, ..., &8600, &8700, 8800.

Diese OS Funktion wird von der Benutzeroberfläche TURBO DESK verwendet.

UMWANDLUNG ALLER 32 BYTE IN 16 BYTE EINTRÄGE

Kurzbeschreibung: Diese OS Funktion wird von der TURBO DESK Benutzeroberfläche benutzt.

Label: DIRWA

ROM-Nummer: C

Startadresse: &FD9E

Einsprungbedingungen: TURBO_X muß einen gültigen Wert enthalten.
Bei TURBO_A bis TURBO_M müssen von den 8 Bytes jehweils die ersten 4 Bytes mit gültigen Werten belegt sein. Bei inaktiven Laufwerken reicht es aus deren Inaktivität im Tagging Byte festzuschreiben.
Es muß mindestens halb so viel freier Erweiterungsspeicher vorhanden sein, wie die Speichermenge, die alle eingelesenen Directorys verbrauchen.

Aussprungbedingungen: Zu jedem eingelesenen Directorys eins Laufwerks existiert mun ein ausgabereifes Directory, das in den Variablen TURBO_A bis TURBO_M beschrieben wird. Außerdem wurde die Variable TURBO_X aktualisiert.

Manipuliert: AF, BC, DE, HL, DE', HL', IY und XH

Beschreibung: Wählt man bei der Benutzeroberfläche TURBO DESK z.B. die Directoy Funktion an, dann werden zuerst alle Directokry eingelesen.

Diese Inhaltsverzeichnisse müßen anschließend für die Bildschrimausgabe aufbereitet werden, dies übernimmt diese OS Funktion. Es wird dabei intensiver Gebrauch der Systemvariablen gemacht. Die Variablen TURBO_A bis TURBO_M, die jeh aus 8 Bytes bestehen müssen bei aktivem Laufwerk in den ersten 4 Bytes gültige Daten enthalten. Es müssen jedoch alle Tagging Bytes dieser Variablen einen Wert enthalten, der den Laufwerksstatus anzeigt. Die Variable TURBO_X muß einen gültigen Wert enthalten, so daß für die aufbereiteten Inhaltsverzeichniße genug Platz ist. Die OS Funktion blendet das zugehörige RAM eines jeden markierten d.h. getaggten Laufwerks ein, konvertiert das Directory in das für die Bildschirmausgabe aufbereitete Format, und speichert diesen aufbereiteten Datenblock im Erweiterungspeicher ab. Die Information darüber wird in den Variablen TURBO_A bis TURBO_M gesichert. Außerdem wird die Systemvariable TURBO_X aktualisiert, sie zeigt nun auf das höchste freie Byte im höchstem freien 16K Erweiterungsblock.

Bitte Beachten: Diese OS Funktion sollte nur dann aufgerufen werden wenn genug RAM vorhanden ist, da sie bei einer zu geringen RAM Menge abbricht.

LÖSCHEN DES BILDSCHIRMSPEICHERS

Kurzbeschreibung: OS Funktion löscht den 16K Bildschirmspeicher.

Label: LESC

ROM-Nummer: C

Startadresse: &C017

Einsprungbedingungen: V-RAM muß bei &C000 beginnen (ist normal).

Aussprungbedingungen: Der Speicherbereich von &C000 bis &FFFF ist mit &00 Bytes beschrieben worden. A=&00 und BC=DE=HL=&0000

Manipuliert: AF, BC, DE, HL, HL' und &4000 Bytes Video-RAM

Beschreibung: Diese OS Funktion entspricht in etwa dem Basic Befehl CLS, denn sie löscht den gesammten Bildschirmspeicher von &C000 bis &FFFF. Dieser Bereich wird mit &00 Bytes gefüllt.

Die Besonderheit dieser OS Funktion ist ihre enorme Geschwindigkeit, pro Byte werden nur 1.5 µs verbraucht. Nach knapp 0.025 Sekunden ist alles vorbei.

Bitte Beachten: Es existiert keine Möglichkeit den 16 KB Bildschirm-Speicher des CPC schneller zu löschen.

LÖSCHEN DER UNTEREN BILDSCHIRMHÄLFTE AB ZEILE 15

Kurzbeschreibung: Beim 64 Zeichen auf 32 Zeilen Format wird die untere Bildschirmhälfte von der 15. Zeile bis zum Bildschirmende gelöscht.

Label: LEDA

ROM-Nummer: C

Startadresse: &C4F5

Einsprungbedingungen: V-RAM muß bei &C000 beginnen (ist normal).

Aussprungbedingungen: untere Bildschirmhälfte gelöscht

Manipuliert: AF, BC, DE, HL, HL' und 18 Zeilen (15-32)

Beschreibung: Diese OS Funktion dient dazu den unteren Teil des Video- RAMs zu löschen (auf &00 zu setzen). Es wird ab der 15. Zeile bis zum Ende des Bildschirms gelöscht (18 Zeilen).

Diese OS Funktion sollte nur verwendet werden wenn das Bildschirmformat auf 64 * 32 gesetzt wurde (64 Zeichen pro Zeile und 32 Zeilen pro Seite). Denn bei anderen Formaten wird nicht am Zeilenanfang mit dem Löschen begonnen.

Da LEDA die FutureOS FILL Funktion verwendet, ist sie praktisch nicht mehr zu beschleunigen.

Diese OS Funktion wird z. B. vom Turbo Desk verwendet, um den Bereich zu löschen, in dem die Directorys dargestellt werden.

Bitte Beachten: Bilschirm muß auf 64 Zeichen bei 32 Zeilen gesetzt sein.

Tabelle generieren um IBM, DATA oder SYSTEM Format Dateien zu LADEN

Kurzbeschreibung: Eine Track, Sektor Tabelle wird generiert. Für IBM, Data oder System Format.

Label: LTAI (IBM Format) // LTAB (DATA Format) // LTAS (SYSTEM Format)

ROM-Nummer: C

Startadresse: &C664 (LTAI) // &C539 (LTAB) // &C710 (LTAS)

Einsprungbedingungen:
HL = 1. Byte des 1. Extents der Datei, 32er DIR
HL' = 1. Byte der Track, Sekror Tabelle (Ziel)

Aussprungbedingungen: Ladetabelle wurde generiert.

Manipuliert: AF, BC, DE, HL, BC', DE' und HL'

Beschreibung: Um eine Datei zu laden muss zuerst eine Ladetabelle generiert werden. Dies übernimmt diese OS Funktion für IBM, System und Data Format.

Dabei muss zuerst das 32er DIRectory eingeblendet werden, dann läd man HL mit der Adresse des Anfangs des ersten Extents der Datei die man später laden will. Außerdem ist HL' mit der Zieladresse zu laden, ab der die zu generierende Tabelle (ent)stehen soll.

Aufbau der Tabelle bei IBM, System oder Data Format:

Die Tabelle beginnt mit einer Tracknummer, dieser folgt die Anzahl der Sektoren, die von diesem Track zu laden sind. Der Sektoranzahl folgen die einzelnen Sektornummern.

Dannach folgt die nächste Tracknummer usw.

- Wenn die Sektoranzahl = &FF ist, dann ist der entsprechende Track komplett zu laden. In diesem Fall folgt auf die Traknummer nur das Byte &FF, aber KEINE Sektornummern. Nach diesem &FF folgt direkt die nächste Tracknummer.

- Ist die Tracknummer = &FF, dann ist die Tabelle zuende. Das Tabellenende wird also durch eine Tracknummer von &FF signalisiert.

Bitte Beachten: HL sollte mit einem korrektem Wert geladen sein, denn ist ab HL kein Extent eines 32er DIRs, so kann dies zu unvorhersehbaren Fehlfunktionen führen.

TABELLE GENERIEREN UM VORTEX FORMAT DATEIEN ZU LADEN

Kurzbeschreibung: Eine Track - Sektor Tabelle wird generiert. Es handelt sich um eine Datei auf VORTEX Format.

Label: LTAV

ROM-Nummer: C

Startadresse: &C7FB

Einsprungbedingungen:
HL = 1. Byte des 1. Extents der Datei, 32er DIR
HL' = 1. Byte der Track, Sekror Tabelle (Ziel)

Aussprungbedingungen: Ladetabelle wurde generiert.

Manipuliert: AF, BC, DE, HL, BC', DE' und HL'
Die 2 Bytes vor HL' werden ebenfalls manipuliert, da sie kurzzeitig als Pseudotabelle verwendet werden.

Beschreibung: Um eine Datei zu laden muss zuerst eine Ladetabelle generiert werden. Diese OS Funktion generiert eine Tabelle für VORTEX Format, dessen Tabellenaufbau von dem der anderen Formate abweicht.

Dabei muss zuerst das 32er DIRectory eingeblendet werden, dann läd man HL mit der Adresse des Anfangs des ersten Extents der Datei die man später laden will.

Außerdem ist HL' mit der Zieladresse zu laden, ab der die zu generierende Tabelle (ent)stehen soll.

Aufbau der Tabelle bei Vortex Format:

Die Tabelle zu Vortex Format hat an sich den selben Aufbau wie Die zum Laden von System oder Data Format.

Der Unterschied liegt darin, daß beim Vortex Format zu jeder Spur auch immer eine Kopfnummer angegeben werden muss. Die Kopfnummer ist im Bit 0 des jehweiligen Trackbytes enthalten.
Rechnerisch wird dazu die Tracknummer (0 bis 80) mit 2 multipliziert und das Kopfbit (0 = oben, 1 = unten) addiert. Für die Ober- bzw. Unter- Seite einer Spur sind also zwei Trackbytes enthalten.
Die physikalische Tracknummer kann durch einfaches rechtsschieben mit z.B. SRL A errechnet werden. Das Carry Bit hat dann die Kopfnummer.

Bitte Beachten: HL sollte mit einem korrektem Wert geladen sein, denn ist ab HL kein Extent eines 32er DIRs, so kann dies zu unvorhersehbaren Fehlfunktionen führen.
ACHTUNG: Die zwei Bytes vor HL' werden manipuliert, eventuell darin abgelegte Daten sind verloren.

VERGLEICHEN ZWEIER EXTENTS IN EINEM DIRECTORY

Kurzbeschreibung: Diese OS Funktion gibt Aufschluß darüber ob zwei Extents der selben Datei angehören. Es werden User, Name und Extension verglichen.

Label: EXCP

ROM-Nummer: C

Startadresse: &C94D

Einsprungbedingungen: DE = &XXX0 = 1. Extent, der zu Vergleichen ist.
HL = &YYY0 = 2. Extent, der zu Vergleichen ist.
Beide Extent-Adressen müssen durch 16 teilbar sein, d.h. die unteren 4 Bit in E und L müssen auf Null gesetzt sein.

Aussprungbedingungen: Z-FLAG gesetzt (Z): beide Extents gehören zur selben Datei.
Z-FLAG geleert (NZ): die Extents sind ungleich, gehören also zu verschiedenen Dateien.
DE und HL (Zeiger auf Extents) bleiben erhalten.

Manipuliert: AF und BC

Beschreibung: Manchmal ist es nötig zwei Extents zu vergleichen, z.B. um zu sehen ob eine Datei noch einen Extent hat. Für solche Aufgaben ist diese OS Funktion gedacht.

Beim Aufruf werden die Adressen der beiden zu vergleichenden Extents in DE und HL übergeben. Aber Achtung: Beide Adressen müssen durch 16 teilbar sein, also der Form &???0 entsprechen. Nun vergleicht die OS Funktion die Usernummer, den Namen und die Extension der Datei. Alle anderen Bytes werden nicht beachtet. Von den 32 Bytes eines Extents werden also nur die ersten 12 verglichen. Stimmen diese 12 Bytes überein, d.h. beide Extents gehören zur selben Datei, dann kehrt die OS Funktion mit gesetzten Zero-Flag zurück. Sind die Extents aber unterschiedlich, so wird das Z-Flag gelöscht.

Die Adressen beider Extents werden erhalten, nur die Register AF und BC werden verändert.

Bitte Beachten: Die Adresse beider Extents muss durch 16 teilbar sein.
Sonst tritt u.U eine Fehlfunktion auf, die beim Aussprung einen falschen Status liefert.
Diese OS Funktion bezieht sich auf ganz normale 32 Byte Extents, wie sie im Inhaltsverzeichnis stehen. Es werden jedoch nur die ersten 12 Bytes verglichen.

INHALTSVERZEICHNISSE NEU SORTIEREN UND WANDELN 32 => 16

Kurzbeschreibung: Alle markierten & manipulierten Inhaltsverzeichnisse werden neu sortiert und anschließen vom 32er in den 16er Modus gewandelt.

Label: ISWS

ROM-Nummer: C

Startadresse: &FDA1

Einsprungbedingungen: RAM Variablen TURBO_A..M müssen korrekt sein.

Aussprungbedingungen: Alle betroffenen DIRs sind neu sortiert und gewandelt worden.

Das untere ROM ist ausgeblendet.

Manipuliert: Alle Register außer AF' und XL. Die RAM Variablen REG16_0..9 usw.
Achtung: Der RAM Bereich von &0000 bis &3FFF ist verändert worden.
Diese 16 KB haben Pufferfunktion.

Beschreibung: Wurde(n) ein(ige) Inhaltsverzeichnis(se) manipuliert, z.B. durch eine Dateioperation wie ERA, REN oder ähnliches, dann befindet sich das (befinden sich die) Directory(s) nicht mehr in sortiertem Zustand. Auch sind die 16er DIRs nicht mehr korrekt.

Diese OS Funktion prüft welche Laufwerke aktiv sind, und welche LWs zusätzlich auch noch verändert wurden (gesetztes Bit 3 in TURBO_A..M).

Bei den Laufwerken die tatsächlich aktiv sind, und deren DIRs verändert wurden, wird diese OS Funktion aktiv. Zuerst wird DIR für DIR erneut sortiert. Danach werden neue 16er DIRs generiert. Nun befinden sich die DIRs im selben Zustand, wie nach ihrem ersten Laden.

Es werden dabei jehweils die alten RAM Blöcke benutzt, so wird kein Speicher verschwendet, eher gewonnen.

Was ist nun ein manipuliertes/verändertes LW/DIR? Der Status eines Laufwerks wird dann auf MANIPULIERT gesetzt, wenn in seinem DIR eine Veränderung durchgeführt wurde. Benennt man z.B. eine Datei mit dem REN Icon um, dann wird der Status des LWs automatisch auf MANIPULIERT gesetzt. So kann diese OS Funktion erkennen, dass es hier was zu tun gibt.

Bitte Beachten: Nach Aufruf der OS Funktion ist das untere ROM ausgeschalten. Die Zeichenausgabe funktioniert nur wenn ein RAM Zeichensatz geladen wird oder wenn man das untere ROM wieder einblendet.
Der Speicherbereich von &0000 bis &3FFF wurde als Puffer verwendet.
Diese 16KB sind also überschrieben worden.

ERWEITERUNGSRAMS TESTEN UND VARIABLEN SETZEN

Kurzbeschreibung: Alle 16 KB Erweiterungs-RAMs werden auf Vorhandensein getestet und die Systemvariablen entsprechend gesetzt.

Label: RANI bzw. RAMI

ROM-Nummer: C

Startadresse: &C8EF (RANI) bzw. &FD35 (RAMI)

Einsprungbedingungen: keine

Aussprungbedingungen: Die 32 System-Variablen XRAM_C4 .. XRAM_FF wurden gesetzt:
Bit 0 = 0 => 16 KB RAM Block existiert NICHT
Bit 0 = 1 => 16 KB RAM Block ist vorhanden

Manipuliert: AF, BC, DE, HL, AF', BC', DE', HL'
Und der RAM-Status (&7FC0, Hauptspeicher, 1. 64 KB aktiv)
RANI: In allen 32 Erweiterungs-RAMs (a 16 KB) ist das erste Byte an Adresse &4000 manipuliert
RAMI: Die E-RAMs sind zwar erhalten, aber &B7C0-&B7DF ist manipuliert

Beschreibung: Beim Hochfahren des OS ist es nötig zu erfahren wieviel Erweiterungs-RAM vorhanden ist, und wo die Blöcke liegen.
Diese OS Funktion testet die 32 Blöcke (je 16 KB) der ersten 512 KB des Erweiterungs-RAMs. Diese Blöcke können zwischen &4000 und &7FFF eingeblendet werden. Im System-RAM existiert für jeden dieser 32 Blöcke ein Byte. Diese OS Variablen werden mit XRAM_?? bezeichnet, wobei ‚??' die Zeichenfolge C4, C5, C6, C7, CC, CD, CE, CF, D4, .. ‚FF enthalten kann. Die OS Variable XRAM_C4 liegt an Adresse &B9D0, die anderen folgen. Alle 32 OS Variablen werden gesetzt.
Dabei bedeutet ein gesetztes erstes Bit, dass dieser 16 KB RAM Block auch wirklich existiert. Dagegen signalisiert das auf Null gesetztes Bit Null daß der Block nicht vorhanden ist.

Das FutureOS spricht das gesammte Erweiterung-RAM in 16K Blöcken an, die zwischen &4000 und &7FFF eingeblendet werden.

Bitte Beachten: Die OS Funktion RANI manipuliert in jedem 16 KB Block des Erweiterungs-RAMs das erste Byte (an Adresse &4000). Die OS Funktion **RAMI** konserviert das gesamte E-RAM, beschreibt aber den Speicher-Bereich von &B7C0-&B7DF.

FREIEN ERWEITERUNGSSPEICHER BERECHNEN, VARIABLEN SETZEN

Kurzbeschreibung: Der freie, unbenutzte Erweiterungsspeicher wird berechnet, die Systemvariablen werden entsprechend gesetzt.

Label: FESB

ROM-Nummer: C

Startadresse: &C998

Einsprungbedingungen: Die 32 System-Variablen XRAM_C4 .. XRAM_FF müssen korrekt sein.

Aussprungbedingungen:
D = Anzahl freier 16 KB Kurzzeitspeicher.
E = &09
BC = &00F7
HL = &B9EF = Zeiger auf XRAM_FF

Manipuliert: AF, BC, DE, HL und u.U. XRAM_??

Beschreibung: Für alle 32 16KB ErweiterungsRAMs existiert eine Systemvariable XRAM_??. Diese Variable gibt Aufschluß darüber ob das entsprechende 16K RAM vorhanden ist, und wie es verwendet wird.
Die Aufgabe dieser OS Funktion ist es nun alle unbenutzten, oder als Kurzzeitspeicher markierten, 16K RAMs zu zählen und die entsprechenden Systemvariablen XRAM_?? zu markieren. Die OS Funktion übergibt in Register D die Anzahl der freien 16 KB Blöcke. Welche Blöcke dies sind, kann man aus den Systemvariablen XRAM_?? lesen. Die Sys-Vars der freien RAMs werden als Kurzzeitspeicher mit &09 markiert.

Bitte Beachten: Die Daten der 32 System-Variablen XRAM_C4 .. XRAM_FF müssen korrekt sein.
Im Memory Map ist nachzulesen welche Bedeutung die Bits der XRAM_?? Variablen haben.

SUCHE ERSTE MARKIERTE DATEI

Kurzbeschreibung: Alle Datei-Tagging-Bytes aller Medien (A bis O) werden nach der ersten markierten Datei durchsucht.

Label: FMD32

ROM-Nummer: C

Startadresse: &FD98

Einsprungbedingungen: Die OS Variablen TURBO_A bis TURBO_M und die Datei-Tagging Bytes müssen korrekt sein.
A < &FF => Normale Funktion: Der Dateistatus der ersten gefundenen Datei wird von ‚markiert' auf ‚benutzt' gesetzt.
A = &FF => Dateistatus wird NICHT verändert!

Aussprungbedingungen:
HL = &0000 => Es wurde keine markierte Datei gefunden. Das Zero-Flag ist gesetzt!
HL > &0000 => Das Zero Flag ist geleert. Und HL zeigt auf die Adresse des ersten Extents im 32er DIR der ersten markierten Datei (Medien A-M). Bei Medium N und O zeigt HL auf den Namen der Datei im M4 SD-Inhaltsverzeichnis-E-RAM.
REG08_1 = A = Medium auf dem sich die markierte Datei befindet. Die Werte 0 bis 14 entsprechen den Medien ‚A' bis ‚O'.
REG08_2 = phys. E-RAM Nummer &C4 ... &FF in dem sich das gepufferte Inhaltsverzeichnis (32er DIR) der markierten Datei befindet. Dieses E-RAM ist bereits von &4000 bis &7FFF eingeblendet. Bei Medium ‚N' oder ‚O' (M4 SD-Karte) ist es das E-RAM für die M4-Erweiterung.
REG16_6 bis REG32_1 = Enthalten den 16 Byte Dateinamen, so wie er im 16er DIR steht (im Format 'A00:FileNameExt').
Bei den SD-Karten Medien ‚N' und ‚O' steht hier eine konvertierte Version des Dateinamens (im Format 'N--:FileNameExt').
FN_LR = &00 / &20, falls die Datei auf Medium 'N' / 'O' gefunden wurde.
Die Laufwerksmotoren wurden eingeschalten.

Wenn A bei Aufruf ungleich &FF war:
Der Datei-Status wird von Tagged/Markiert auf Used/Benutzt gesetzt, eine soche Datei erscheint auf dem Bilschirm durchgestrichen. Ruft man FMD32 nochmals auf, so wird die nächste markierte Datei gesucht.

Manipuliert: Alle Register außer IY. RAM Status, Datei-Tagging Bytes,
Motor Status, REG08_1..2, REG16_6..REG32_1

Beschreibung: Dieses Unterprogramm dient dazu in allen aktiven Medien nach der ersten markierten Datei zu suchen. Dabei wird von Laufwerk A bis zu SD Karte ‚O' gesucht. Beim Einsprung müssen die Systemvariablen in Ordnung sein.
Die OS Funktion übergibt in HL die Adresse des ersten Extents der ersten markierten Datei. Hat das Register HL den Wert &0000 und das Zero Flag ist gesetzt, dann wurde keine markierte Datei gefunden.

Ansonsten wird im Akku und in REG08_1 das Medium, auf dem sich die markierte Datei befindet, übergeben. Die Werte 0 bis 14 entsprechen den Laufwerken bzw. Medien ‚A' bis ‚O'.
Außerdem wird in REG08_2 die physikalische E-RAM Nummer von &C4 bis &FF übergeben, in dem sich das gepufferte Inhaltsverzeichnis (32er) der markierten Datei befindet. Dieses E-RAM wurde bereits zwischen &4000 und &7FFF eingeblendet.
FMD32 startet die LW-Motoren, wenn ein Laufwerk markiert ist. Trozdem ist die restliche Hochlaufzeit abzuwarten, oder zumindest solange mit dem Laden der Datei zu warten bis das LW bereit (Ready) ist.

Bitte Beachten: Systemvariablen müssen unbedingt korrekt sein, da sonst korrupte Daten übergeben werden. Fehlfunktion.
Achtung: Hat Register A vor Aufruf der OS Funktion den Wert &FF, so wird der Status der gefundenen Datei NICHT verändert. Bei nochmaligem Aufruf von FMD32 würde man die selbe Datei nochmals finden!

BLOCK-BELEGUNGS-TABELLE DATA/IBM/SYSTEM/VORTEX GENERIEREN

Kurzbeschreibung: Die Blockbelegungstabelle eines Laufwerks wird generiert, aber nur Data, IBM, System oder Vortex Format. Das DIR darf unsortiert sein.

Label: BBTG (LW in XL), BBTH (LW in A)

ROM-Nummer: C

Startadresse: &FD95 (BBTG), &FD53 (BBTH)

Einsprungbedingungen: A bzw. XL = Laufwerksnummer von 0 bis 7.
Inhaltsverzeichnis des Laufwerks muß eingelesen sein, System Variablen sollten korrekt sein.

Aussprungbedingungen: Blockbelegungstabelle wurde von &B600 bis &B6FF generiert.
Block &00 (Adr. &B600) wird nie beachtet.
XL = Laufwerk, dessen Tabelle generiert wurde.
XH = Laufwerks-Tag-Byte des Laufwerks aus A bzw. XL.
RAM des 32er DIRs des LWs aus A bzw. XL ist eingeblendet.

Manipuliert: AF, BC, DE, HL, AF', (XL), XH und der RAM Status

Beschreibung: Wenn man eine Datei von Hand speichern will, dann sollte man wissen welche Blöcke auf Diskette noch frei sind.

Um herauszufinden welche Blöcke auf einer Disk mit Data, IBM, System oder Vortex Format noch frei sind benutzt man diese OS Funktion.

Im Akku (Einsprung BBTH) bzw. IX low (Einsprung BBTG) übergibt man die Nummer des Laufwerks, auf das die Datei geschrieben werden soll. Die OS Funktion generiert zwischen &B600 und B6FF eine Blockbelegungstabelle. In Adresse &B601 ist der Status des Blocks &01 angegeben, in &B602 der Status von Block &02, ... und so weiter bis zum letzten Block. Enthält ein Byte den Wert &00, dann ist der entsprechende Block noch frei. Wenn dagegen in einem Byte der Wert &FF steht, dann ist der Block bereits von einer anderen Datei belegt.

Außerdem ist beim Aussprung das RAM des 32er DIRs des entsprechenden LWs eingeblendet.

Das Inhaltsverzeichniss, dessen BBT generiert wird, muß nicht sortiert sein.

ACHTUNG: Die Blöcke des DIRs werden nicht als benutzt markiert! s.u.

Die verschiedenen Formate eines Laufwerkes haben eine unterschiedliche Blockanzahl, und das DIR belegt andere Blöcke. Bei Vortex hat ein Block nicht 1 KB sondern 4 KB.
LW mit Data Format hat 180 Blöcke, DIR in Block 0, 1.
LW mit IBM Format hat 156 Blöcke, DIR in Block 0, 1.
LW mit System Format hat 171 Blöcke, DIR in Block 0, 1.
LW mit Vortex Format hat 177 4K-Blöcke, DIR in Block 0.

Bitte Beachten: Man kann nur Blockbelegungstabellen für Laufwerke generieren, deren DIR ordnungsgemäß eingelesen wurde.

ACHTUNG: Die Blöcke des DIRs werden nicht als benutzt markiert! Dies ist einfach von Hand zu machen:
Bei Data, IBM und System Format sind die Blöcke &00 und &01 für das DIR reserviert. Beim Vortex-Format belegt das DIR nur den Block &00.

Das Register XH enthält das Laufwerks-Tag-Byte des Laufwerks aus XL, dies enthält auch die Format-Information für das LW.

LADEN EINER DURCH IHREN NAMEN DEFINIERTEN DATEI

Kurzbeschreibung: Eine Datei, deren Name und Medium bekannt ist wird geladen.

Label: LADE_N

ROM-Nummer: C

Startadresse: &FD5C

Einsprungbedingungen: Das Inhaltsverzeichnis in dem sich die zu ladende Datei befindet muß eingelesen sein (Medium aktiviert, DIR-Icon wurde benutzt).
A = Medium (0..14) von dem die Datei geladen werden soll
Ist das 8. Bit gesetzt, so werden Lade- und Start-Adresse des Headers ignoriert
DE = Zeiger auf User-Nummer (1 Byte), Name (8 Bytes) und Extension (3 Byte) der Datei.
Oder DE zeigt auf M4 Namen.
Der Name muß sich innerhalb einer 256 Bytes Page befinden.

Falls die zu ladende Datei keinen Header hat, oder der Header ignoriert werden soll, sind außerdem folgende Daten anzugeben:
REG08_4 = Ladeart (0,1,2,3) - siehe OS Funktion(en) ‚LADEN'
Für die Ladearten 2 und 3 muss folgendes angegeben werden:
REG16_3 = Ladeadresse im RAM bzw. E-RAM, 16 Bit
Für die Ladeart 3 muss folgendes angegeben werden:
AKT_RAM = physikalische E-RAM Nummer, &C4, &C5 .. &FF

Aussprungbedingungen: Im Akku wird ein Infobyte übergegen, daß Aufschluß über den Verlauf der Operation gibt.
A = &00 => Momentan ist überhaupt kein Inhaltsverzeichnis eingelesen, deshalb kann auch keine Datei geladen werden.
A = &01 => Das Medium von dem die Datei geladen werden soll ist nicht markiert/aktiv, die Datei kann nicht geladen werden.
A = &02 => Die Datei existiert nicht im Inhaltsverzeichnis des angegebenen Mediums --> Die E-RAM Konfiguration ist unbekannt!
A = &FF => Die Datei sollte ordnungsgemäß geladen sein, es ist aber durchaus sinnvoll noch die 7 Resultbytes ab FDC_RES zu testen, falls man von einem Floppy Disk Laufwerk geladen hat.
(REG_PC+1) = Code von 0 bis 14 des Mediums (A bis O, LW, HD20, MM, SD-Karte) von dem die Datei geladen wurde.

Manipuliert: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY und die RAM-Variablen REG08_0-3,5,6, REG16_0,1, REG_PC. Außerdem die RAM Bereiche von &B000 bis &B7FF und von &BC00 bis &BC7F, der E-RAM-Status, der FDC, der Bildschirm MODE und der Status der Motoren der Laufwerke, die Datei-Markierungs-Bytes. Bei einem Fehler ist nicht bekannt welches E-RAM von &4000 bis &7FFF eingeblendet ist.

Beschreibung: Diese OS Funktion dient dazu eine beliebige Datei zu laden. Die Datei wird durch ihr Quell-Medium, ihre User-Nummer (nur A-M), ihren Namen und ihre Extension

eingeutig identifiziert. Dabei ist es egal ob die Datei markiert ist oder nicht, ihr Status wird nicht beeinflusst.

Das Medium wird in A angegeben, ist das 8. Bit gesetzt, so wird ein eventueller Dateiheader ignoriert. Ansonsten entscheidet der Header wohin die Datei geladen wird. Ohne Datei-Header (oder mit ignoriertem Header) entscheiden folgende RAM-Variablen, an welche Adresse, in welchen RAM Block, und auf welche Art die Datei geladen wird.

REG08_4 = Ladeart: Die Ladeart &02 läd die Datei in den Hauptspeicher, die Ladeart &03 läd die Datei ins Erweiterungs-RAM.

REG16_3 = Ladeadresse im RAM bzw. E-RAM, 16 Bit

AKT_RAM = physikalische E-RAM Nummer, &C4, &C5 .. &FF

Wurde die Datei korrekt geladen, so kehrt die OS Funktion mit &FF im Akku zurück, andernfalls ist die Fehlerursache im Akku verschlüsselt (RAM Konfiguration undefiniert). Es empfiehlt sich immer auch die 7 Result-Bytes des FDCs zu testen. Diese stehen ab FDC_RES im RAM.

(REG_PC+1) enthält die Nummer des Mediums, von der die Datei geladen wurde. Dabei entsprechen die Nummern 0 bis 14 den Medien von A bis O.

Bitte Beachten: Wurde die Datei korrekt geladen, so kann man sicherheitshalber die 7 Resultbytes ab FDC_RES testen.

LADEN EINER MARKIERTEN DATEI IN DEN HAUPTSPEICHER AB &0000

Kurzbeschreibung: Eine Datei wird ab Adresse &0000 in den Hauptspeicher geladen. (Adresse &0000 ist Standart).

Label: LADEN

ROM-Nummer: C

Startadresse: &FD8F

Einsprungbedingungen: REG08_3 mit Wert &00 füllen für diese Ladeart.
Systemvariablen müssen korrekt sein, ein DIR muss eingelesen sein, und eine Datei muss in den Datei-Tagging-Bytes markiert sein.
A < &FF => normale Funktion
A = &FF => Dateistatus wird NICHT verändert!

Aussprungbedingungen: Datei wurde ab der Adresse &0000 in den Hauptspeicher geladen.
Ihr 128 Byte Header steht ab &BC00 im RAM.
(REG_PC+1) = Medium (0..12) von dem die Datei geladen wurde.

Manipuliert: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY und die RAM-Variablen REG08_0,1,5, REG16_0, REG_PC. Außerdem die RAM Bereiche von &B000 bis &B7FF und von &BC00 bis &BC7F, der RAM-Status, der FDC, die Datei-Tagging Bytes und der Motor Status.

Beschreibung: Diese OS Funktion dient dazu eine Datei in den Hauptspeicher zu laden. Wurde durch die Benutzeroberfläche, oder von Hand, eine Datei markiert (Datei-Tagging-Bytes), dann kann sie durch diese OS Funktion geladen werden. Die Datei wird automatisch an die Standart-Adresse &0000 geladen. Man sollte darauf achten, daß die Datei 40K (maximal 44 KB) nicht überschreitet, da sonst wichtige Teile des OS überschrieben werden. Siehe Memory-Map in: '#EQU-API.DEU'. Der Datei-Header liegt ab &BC00 im RAM.
Bei einer Fehlfunktion z.B. keine Datei markiert, gibt die OS Funktion eine Fehlermeldung aus und springt in die Benutzeroberfläche zurück.

Bitte Beachten: Es sollte ein DIR eingelesen & eine Datei markiert sein. Maximale Dateigröße 40 (44) KB, dann ist das RAM komplett verbraucht.

Achtung: Hat Register A vor Aufruf der OS Funktion den Wert &FF, so wird der Status der geladenen Datei NICHT verändert, beim nächsten Aufruf einer Dateifunktion wird dann die selbe Datei nochmals behandelt!

LADEN einer markierten DATEI in den Hauptspeicher ab beliebiger Adresse

Kurzbeschreibung: Eine Datei wird ab beliebiger Adresse in den Hauptspeicher geladen.

Label: LADEN

ROM-Nummer: C

Startadresse: &FD8F

Einsprungbedingungen: REG08_3 mit Wert &02 füllen für diese Ladeart
REG16_1 = Zieladresse der Datei
Systemvariablen müssen korrekt sein, ein DIR muss eingelesen sein, und eine Datei muss in den Datei-Tagging-Bytes markiert sein.
A < &FF => normale Funktion
A = &FF => Dateistatus wird NICHT verändert!

Aussprungbedingungen: Datei wurde ab der Adresse aus REG16_1 in den Hauptspeicher geladen. Ihr 128 Byte Header steht ab &BC00 im RAM.
(REG_PC+1) = Medium (0..12) von dem die Datei geladen wurde.

Manipuliert: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY und die RAM-Variablen REG08_0,1,5, REG16_0, REG_PC. Außerdem die RAM Bereiche von &B000 bis &B7FF und von &BC00 bis &BC7F, der RAM-Status, der FDC, die Datei-Tagging Bytes und der Motor Status.

Beschreibung: Diese OS Funktion dient dazu eine Datei in den Hauptspeicher zu laden. Die Zieladresse ist dabei frei wählbar, sie wird in der Speicherstelle REG16_1 angegeben. Wenn man eine Datei an die Adresse &C000 läd, also in den Bildschirmspeicher, dann kann die Datei maximal 60 KB groß sein. Denn sobald beim LADEN die Adresse &FFFF überschritten wird, wird ab Adresse &0000 weitergeladen. Man sollte sich im klaren sein welche Speicherbereiche das OS bzw. die Benutzeroberfläche belegt. Der 128 Byte Datei-Header wird ab &BC00 ins RAM geschrieben.

Bitte Beachten: Es sollte auch ein DIR eingelesen & eine Datei markiert sein. Maximale Dateigröße 40(60) KB, dann ist das RAM komplett verbraucht.

Achtung: Hat Register A vor Aufruf der OS Funktion den Wert &FF, so wird der Status der geladenen Datei NICHT verändert, beim nächsten Aufruf einer Dateifunktion wird dann die selbe Datei nochmals behandelt!

Laden einer mark. Datei in Erweiterungsspeicher ab Block &C4, Adr. &4000

Kurzbeschreibung: Eine Datei wird in Block &C4 ab Adresse &4000 in den Erweiterungsspeicher geladen. (Standart).

Label: LADEN

ROM-Nummer: C

Startadresse: &FD8F

Einsprungbedingungen: REG08_3 mit Wert &01 füllen für diese Ladeart.
Systemvariablen müssen korrekt sein, ein DIR muss eingelesen sein, und eine Datei muss in den Datei-Tagging-Bytes markiert sein.
A < &FF => normale Funktion
A = &FF => Dateistatus wird NICHT verändert!

Aussprungbedingungen: Datei wurde ab Block &C4 und Adresse &4000 in den Erweiterungsspeicher geladen.
(REG_PC+1) = Medium (0..12) von dem die Datei geladen wurde.

Manipuliert: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY und die RAM-Variablen REG08_0,1,5, REG16_0, REG_PC. Außerdem der RAM Bereich von &B000 bis &B7FF, der RAM-Status, der FDC, die Datei-Tagging Bytes und der Motor Status.

Beschreibung: Diese OS Funktion dient dazu eine Datei in den Erweiterungsspeicher zu laden. Die Datei wird dabei automatisch ab dem ersten 16K Erweiterungsblock &C4 und der Adresse &4000 in den Erweiterungsspeicher geladen. Falls ein Dateiheader existiert, bleibt er vor der eigentlichen Datei (ab der Zieladresse) erhalten.

Der User sollte sich im klaren sein ob das Erweiterungs-RAM, in das er laden will, überhaupt existiert d. h. eine Speichererweiterung vorhanden ist. Darüber geben die Speicherstellen XRAM_C4 .. XRAM_FF Auskunft.

Natürlich muß vor Aufruf der OS Funktion auch eine Datei markiert sein, da sonst eine Fehlermeldung ausgegeben wird und die OS Funktion in die Benutzeroberfläche springt.

Bitte Beachten: Es sollte auch ein DIR eingelesen & eine Datei markiert sein. Maximale Dateigröße 512 KB, dann spätestens ist das RAM komplett verbraucht.

Achtung: Hat Register A vor Aufruf der OS Funktion den Wert &FF, so wird der Status der geladenen Datei NICHT verändert, beim nächsten Aufruf einer Dateifunktion wird dann die selbe Datei nochmals behandelt!

LADEN einer markierten DATEI IRGENDWO in den ERWEITERUNGSPEICHER

Kurzbeschreibung: Eine Datei wird in den Erweiterungsspeicher geladen, dabei ist der Startblock und die Startadresse frei wählbar.

Label: LADEN

ROM-Nummer: C

Startadresse: &FD8F

Einsprungbedingungen: REG08_3 mit Wert &03 füllen für diese Ladeart.
REG16_1 = Zieladresse der Datei, durch 256 teilbar z.B. &2300, &7800, &C000.
AKT_RAM = RAM-Block ab dem geladen wird.
Systemvariablen müssen korrekt sein, ein DIR muss eingelesen sein, und eine Datei muss in den Datei-Tagging-Bytes markiert sein.
A < &FF => normale Funktion
A = &FF => Dateistatus wird NICHT verändert!

Aussprungbedingungen: Datei wurde ab Block aus AKT_RAM und Adresse aus REG16_1 in den Erweiterungsspeicher geladen.
(REG_PC+1) = Medium (0..12) von dem die Datei geladen wurde.

Manipuliert: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY und die RAM-Variablen REG08_0,1,5, REG16_0, REG_PC. Außerdem der RAM Bereich von &B000 bis &B7FF, der RAM-Status, der FDC, die Datei-Tagging Bytes und der Motor Status.

Beschreibung: Diese OS Funktion dient dazu eine Datei in den Erweiterungsspeicher zu laden. Der Zielblock und die Zieladresse sind dabei frei wählbar. Falls ein Dateiheader existiert, bleibt er vor der eigentlichen Datei (ab der Zieladresse) erhalten.

Siehe auch vorige OS Funktionen, obige Restriktionen gelten auch in diesem Fall.

Beim laden einer größeren Datei (einige 100 KB) muß zweifelsfrei der Erweiterungsspeicher (E-RAM) benutzt werden. Um etwas in das E-RAM zu laden gibt man in der Speicherstelle AKT_RAM die physikalische E-RAM-Nummer &C4,..&FF an, in REG16_1 gibt man die Zeiladresse von &0000 bis &7F00 an. Adressen müssen durch 256 teilbar sein, da sonst Lade-Fehler auftreten werden.

Wenn man als Startadresse einen Wert zwischen &0000 und &3F00 angibt, dann wird anfangs in ein Stück des Hauptspeichers geladen.

Bitte Beachten: Es sollte auch ein DIR eingelesen & eine Datei markiert sein. Maximale Dateigröße 512 KB, dann spätestens ist das E-RAM komplett verbraucht. 4 MB?
Achtung: Hat Register A vor Aufruf der OS Funktion den Wert &FF, so wird der Status der geladenen Datei NICHT verändert, beim nächsten Aufruf einer Dateifunktion wird dann die selbe Datei nochmals behandelt!

SICHERN ALLER REGISTER & EINSPRUNG IN DEN FutureOS MONITOR

Kurzbeschreibung: Bei Aufruf dieser OS Funktion werden alle Register des Z80 Prozessors gesichert. Es erfolgt KEIN Rücksprung.

Label: CARET

ROM-Nummer: C

Startadresse: &FD62

Einsprungbedingungen: - KEINE -

Aussprungbedingungen: Es erfolgt KEIN Rücksprung!

Manipuliert: RAM Variablen: REG_AF1, REG_BC1 .. REG_R, REG_I werden mit Prozessor-Register-Inhalten geladen.

Beschreibung: Diese OS Funktion ist ein Teil des FutureOS Monitors.

Wird sie angesprungen, dann wird der Inhalt ALLER Register (abgesehen von PC) in die RAM Variablen REG_AF1, REG_BC1 ... REG_R, REG_I geschrieben. Der Inhalt des R Registers wird so korregiert, dass ein unterbrochenes Programm dies nicht feststellen können sollte.

Diese OS Funktion kehrt nicht zurück, sondern springt direkt in den FutureOS Monitor ein, von dort aus kann man das zu testende Programm wieder anspringen.

Will man eine OS Funktion testen, so kann man vor dem Aufruf dieser OS Funktion die Adresse von CARET auf den Stack legen, und sie durch JP anspringen. Wenn die zu testende OS Funktion mit RET endet, dann kommt man automatisch nach CARET und in den Monitor.

Bitte Beachten: Auch das R Register wird korrekt behandelt. Es erfolgt KEIN Rücksprung!

DATEI AUS DIR TILGEN, EINTRAG AUF &E5 SETZEN

Kurzbeschreibung: Bevor man eine Datei speichert, sollte man ihre alte Version von Diskette tilgen. Diese Funktion wird von dieser OS Funktion erfüllt.

Label: EWEG

ROM-Nummer: C

Startadresse: &FD7A

Einsprungbedingungen:
REG08_1 = LW, aus dessen DIR die Datei entfernt werden soll.
Ab der RAM-Variable REG16_8 stehen 12 Bytes, die User (1 Byte), Name (8 Bytes), Extension (3 Bytes) der Datei darstellen, die eliminiert werden soll.

Aussprungbedingungen:
Datei wurde aus dem Inhaltsverzeichnis entfernt.
Das Inhaltsverzeichnis ist nun unsortiert!
TURBO_? des LWs wurde als MANIPULIERT (Bit 3) markiert.
Das 32er DIR des LWs wurde eingeblendet, aber nicht im AKT_RAM vermerkt.

Manipuliert: AF, BC, DE, HL, XH und der RAM-Status.

Beschreibung: Will man eine Datei auf Diskette speichern, so ist möglicherweise schon eine Datei mit der selben Bezeichnung vorhanden.

Diese OS Funktion entfernt eine durch User, Name und Extension definierte Datei aus dem DIR eines bestimmten Laufwerks.

Beim Aufruf werden Laufwerk und Dateibezeichnung angegeben. Nach entfernen der Datei aus dem DIR ist selbiges natürlich unsortiert, es sind einige mit &E5 markierte Einträge vorhanden, dies war einmal die Datei.

Bitte Beachten: Diese OS Funktion markiert in TURBO_? des behandelten LWs das Bit 3, das Inhaltsverzeichnis hat also den Status: MANIPULIERT, und wird von der OS Funktion ISWS erkannt und auf Diskette geschrieben.

TABELLE FREIER BLÖCKE GENERIEREN (für SAVE)

Kurzbeschreibung: Will man eine Datei speichern, dann braucht man dazu freie Blöcke auf Disk. Diese OS Funktion generiert eine Tabelle aus freien Blöcken, in die noch Daten gespeichert werden können. Nur Disk!

Label: BBTT

ROM-Nummer: C

Startadresse: &FD86

Einsprungbedingungen: BBTG(!) Tabelle muß ab &B600 vorhanden sein!
B = Anzahl gesuchter, leerer Blöcke 1..255.
C' = maximale Anzahl der Blocks pro Disk bezogen auf das aktuelle Format.
HL = Zieladresse, ab der die Blocktabelle freier Blöcke generiert wird.
HL sollte am Anfang einer Page liegen, es müssen alle Blöcke in diese Page passen. &XX00 !!!

Aussprungbedingungen:
Carry-Flag auf 1 gesetzt => Datei ist zu GROSS.
Carry-Flag mit 0 geleert => alles in Ordnung, die Datei passt noch auf die Disk. =>
HL = Ende der Blocktabelle.
Tabellen-Ende ist durch &00 markiert.

Manipuliert: AF, B, L und HL'

Beschreibung: Will man eine Datei auf Diskette sichern, so wird sie auf verschiedene Blocks verteilt. Diese OS Funktion dient dazu eine bestimmte Anzahl freier Blöcke zu bestimmen, und in eine Tabelle zusammenzufassen.
Bevor man diese OS Funktion benutzt muß man eine Blockbelegungstabelle generieren, in der vermerkt ist, welche Blöcke überhaupt noch frei sind. Dazu verwendet man BBTG, BlockBelegungsTabelle muß ab &B600 generiert werden. Ist Block &01 ein DIR Block, so ist er von Hand als besetzt zu markieren (DATA, SYSTEM, IBM). Block &00 wird NICHT beachtet. Anschließend kann man diese OS Funktion verwenden. Im Register B wird die Anzahl gesuchter Blocks übergeben. Im Data, System und IBM Format ist jeder Block 1KB groß, aber im Vortex Format hat ein Block bereits 4 KB. Im Register C' (nicht C) ist die Gesammt-Anzahl freier Blöcke des aktuellen Formates anzugeben, dies ist von Format zu Format verschieden, und hängt auch von der Anzahl benutzter Spuren ab. In HL wird schließlich die Zieladresse angegeben, ab der die Tabelle freier Blocke generiert werden soll. Es ist darauf zu achten, dass die ganze Tabelle innerhalb einer Page Platz findet, HL sollte also mit &XX00 geladen werden.

Nun wird die OS Funktion angesprungen, die Tabelle wird generiert, und der Erfolg der Operation wird im Carry Flag angezeigt. Bei gesetztem Carry passt die Datei NICHT auf die Diskette. Ist das Carry dagegen geleert, dann passt die Datei auf die Disk, und HL zeigt auf das Ende der Tabelle freier Blöcke. Das Tabellenende wird durch ein &00 Byte markiert.

Bitte Beachten: Gibt man beim Einsprung die Blockanzahl im Register B mit &00 an, so bedeutet dies dass 256 Blöcke gesucht werden. B sollte also einen Wert größer Null haben.

Vor Aufruf dieser OS Funktion ist unbedingt eine Gesammt-Blocktabelle ab &B600 mittels BBTG zu generieren! Nur für Disk geeignet, nicht HD!

ANZAHL FREIER EINTRÄGE eines InHaltsVerzeichnisses ermitteln (DISK)

Kurzbeschreibung: Die Zahl freier 32 Byte DIR Einträge wird bestimmt, dies ist wichtig um Dateien sichern zu können. Nur für Disketten.

Label: EFED

ROM-Nummer: C

Startadresse: &FD83

Einsprungbedingungen:
REG08_1 = Laufwerk, von 0 bis 7, dessen DIR untersucht werden soll.
Systemvariablen müssen korrekt sein.
DIR darf unsortiert sein.

Aussprungbedingungen:
BC = &0020
E = Anzahl freier 32B Einträge des LWs.
HL = Start des 32er DIRs des LWs aus REG08_1.
32er DIR des LWs ist eingeblendet, aber die Variable AKT_RAM wird NICHT verändert.

Manipuliert: AF, BC, DE, HL und der RAM Status.

Beschreibung: Will man eine Datei auf Diskette sichern, so muß auch im Inhaltsverzeichnis ein entsprechender Eintrag gemacht werden. Große Dateien brauchen mehrere Einträge. Diese OS Funktion ermittelt nun wieviele freie/leere 32 Byte Einträge in Inhaltsverzeichnis eines bestimmten Laufwerks noch frei sind.
In der RAM-Variable REG08_1 wird das LW angegeben, um dass es sich handelt. Nun wird die OS Funktion aufgerufen.
Die OS Funktion übergibt im Register E die Anzahl freier 32 Byte Einträge. Außerdem zeigt HL auf den Start der 32er DIRs der LWs. Das DIR RAM des LWs ist eingeblendet.

Bitte Beachten: Beim Aussprung erhält BC den Wert &0020, dies ist wichtig, wenn man im Anschluß an diese OS Funktion EGEN aufrufen will.
Spaart immerhin 3 µs.
Die RAM-Konfig wird nicht in AKT_RAM vermerkt.
ACHTUNG: EFED ist nur für Disketten zuständig, für HD-Partitionen bitte die OS Funktion EFED_HD aus ROM A verwenden.

Inhaltsverzeichnis-Eintrag aus Tabelle FREIER BLÖCKE GENERIEREN (SAVE)

Kurzbeschreibung: Aus einer Tabelle freier DIR-Blöcke werden DIR-Einträge generiert, und sofort ins DIR eingetragen. NUR DISKETTE!

Label: EGEM bzw. EGEN (BC muss &0020 sein!)

ROM-Nummer: C

Startadresse: &FD80 (EGEM) bzw. &FD7D (EGEN)

Einsprungbedingungen:
A = Anzahl zu generierender 32 Bytes Einträge.
BC = &0020 (NUR bei Einsprung in EGEN),
DE = Start Tabelle freier Blöcke (von BBTT).
HL = Start 32er DIR des Ziel-Laufwerks.
HL'= 12 Bytes: User(1),Name(8),Extension(3) der zu sichernden Datei, OHNE Page-übertritt!
YL = &00 => 1 KB Blockgröße (Data, System, IBM) bzw.
YL > &00 => 4 KB Blockgröße (Vortex Format). Das 32er DIR RAM muß bereits eingeblendet sein.
Im DIR müssen genügend freie Einträge zur Verfügung stehen (=> EFED).
Das DIR darf unsortiert sein, freie Einträge werden herausgepickt.

Aussprungbedingungen: Inhaltsverzeichnis-Eintrag wurde generiert.

Manipuliert: AF, BC, DE, HL, AF', BC', DE', HL' und DIR.

Beschreibung: Diese OS Funktion erlaubt es, im RAM - gepufferten 32er Directory Einträge zu installieren, es handelt sich also um einen Eintrags-Generator. Dies ist z.B. beim sichern einer Datei nötig.

Zum Aufruf der OS Funktion müssen: Anzahl 32 Byte-Einträge (A), der Start der Tabelle freier Blöcke (DE), der Start des bereits eingeblendeten 32er DIRs des LWs (HL), User,Name,Extension der Datei (HL') und die Blockgröße 0/&FF (YL) bekannt sein.

Nach verlassen der OS Funktion ist der Eintrag im DIR vermerkt. Es darf keinesfalls mit korrupten Parametern Eingesprungen werden.

Das Inhaltsverzeichnis des LWs ist nun bereit um es nach einem Sortiervorgang (mittels OS Funktion ISWS) auf Diskette zu schreiben.

Bitte Beachten: Es stehen zwei Einsprünge zu Verfügung: EGEM und EGEN, dabei ist EGEN um 3 µs schneller, braucht aber bereits den Wert &0020 im Registerpaar BC. Beim Einsprung in EGEM kann BC einen beliebigen Wert haben.
ACHTUNG: EGEM bzw. EGEN sind nur für Disketten zuständig, für HD-Partitionen bitte die OS Funktion EGEN_HD aus ROM A verwenden.

PROGRAMM VERLASSEN UND RÜCKKEHR INS DESKTOP

Kurzbeschreibung: Dies ist ein Einsprung / Rücksprung ins OS für Programme, die die Icons intakt gehalten haben.

Label: FORA

ROM-Nummer: C

Startadresse: &FD77

Einsprungbedingungen: Die Icons des Desktop und das Bildschirmformat (64 * 32) müssen noch erhalten sein.

Aussprungbedingungen: - KEINE -

Manipuliert: irrelevant

Beschreibung: Will ein kleineres Programm wieder ins Desktop bzw. OS zurückkehren, so kann es diesen Einsprung verwenden, wenn die Icons in der oberen Bildschirmhälfte noch intakt sind, und wenn das Bild noch 64 Zeichen auf 32 Zeilen enthält.

Folgende Aktionen / Restaurationen werden ausgeführt:
* Modus 2 schalten
* beide ROMs einblenden
* 1. Seite der aktuellen DIRs anzeigen, falls vorhanden.
* Sprung nach KLICK in ROM D und damit ins Desktop.

Bitte Beachten: Es erfolgt kein Rücksprung, es sei denn man präpariert zuvor die Variablen des OK Icons und der User klickt irgendwann OK an.

(TEILWEISE) LADEN EINER DATEI IN KURZ-ZEIT-SPIECHER

Kurzbeschreibung: Eine Datei wird, wenn möglich komplett, in die freien 16K KZS geladen.

Label: TEILA (erstes Laden) // TEILB (Rest(e) nachladen)

ROM-Nummer: C

Startadresse: &FD74 (TEILA) // &FD71 (TEILB)

Einsprungbedingungen:
TEILA: XRAM_??, Disk und LW System-Variablen korrekt!
TEILB: REG08_2,4, (REG_PC+1), REG_IY, REG_SP unverändert lassen!!!

Aussprungbedingungen: gilt für TEILA und TEILB:
A = &00 => Die Datei wurde komplett in KZS geladen.
A = &F0 => Die Datei wurde teilweise in die KZS geladen, alle KZS sind voll beladen, Rest mittels TEILB nachladen.
A = &FF => Keine markierte Datei gefunden
REG08_2 = 32er DIR E-RAM-Block der restlichen Datei
REG_IY = Start Dateirest im 32er DIR, falls ein Rest zu laden ist.
REG_PC+1 = Medium 0-12 (Floppy, HD, MM) von dem geladen wurde.

Manipuliert: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY, außerdem die RAM-Variablen: REG08_0...5, REG16_0, 1, REG16_6...REG32_1 (nur TEILA), AKT_RAM, REG_IY, REG_SP, REG_PC und XRAM_C4..FF. Außerdem der RAM-Status, der Motor-Status und die FTB(File Tagging Byte)s.

Beschreibung: Diese OS Funktion ermöglicht es, eine beliebig lange, markierte Datei in die Kurzzeit-Speicher E-RAM Blöcke zu laden. Reicht die Anzahl vohandener KZS (Kurzzeit-Speicher) nicht aus, so wird die Datei nur soweit geladen bis alle KZS Blöcke beladen sind. Der Dateirest kann anschließend nachgeladen werden, notfalls in mehreren Schritten.
Will man eine (lange) Datei z.B. auf dem Bildschirm anzeigen, so passt sie unter Umständen nicht ganz in den Speicher, um dieses Problem zu lösen wurde TEILA/TEILB entwickelt. Zuerst markiert man die zu ladende Datei in den Datei-tagging-Bytes, nun wird TEILA aufgerufen. Alle freien 16K Erweiterungsblocks werden als KZS markiert, und die Datei wird Block für Block in den Speicher geladen.
Passt die Datei ganz in den Speicher, so kehrt die OS Funktion mit A=&00 zurück. Es muß also kein Dateirest nachgeladen werden. Wurde keine markierte Datei gefunden, so hat A den Wert &FF. War die Datei zu groß um sie auf einmal zu laden, so wurden alle KZS mit dem Anfang der Datei beladen, und die OS Funktion kehrt mit A=&F0 zurück. Nun ist dafür Sorge zu tragen, daß die RAM-Variablen REG08_2,4, REG_IY, REG_PC+1 und REG_SP unverändert bleiben, nur dann kann der Dateirest mit TEILB nachgeladen werden. Die Aussprungbedingungen von TEILB entsprechen denen von TEILA. Unter Umständen ist noch ein weiterer Dateirest zu laden, dies hängt von der Gesammtlänge der Datei ab.

Bitte Beachten: Um TEILB korrekt aufzurufen müssen die RAM-Variablen REG08_2, _4, REG_IY, REG_SP und REG_PC+1 unverändert bleiben!

(TEILWEISE) SICHERN EINER DATEI AUS DEM KURZ-ZEIT-SPIECHER (DISK)

Kurzbeschreibung: Eine Datei wird (teilweise) aus dem KZS gesichert.

Label: TEISI (anfangs) und TEISK (bei Dateirest)

ROM-Nummer: C

Startadresse: &FD56 (TEISI), &FD59 (TEISK)

Einsprungbedingungen:

a.) TEISI:
A = Laufwerk, auf das gespeichert werden soll: 0..7 (keine HD, MM !! – stimmt das noch???).
REG_BC1 = Blockanzahl der Datei insgesammt, formatabhängig.
Ab REG16_8 stehen 12 Bytes, die User(1), Name(8) und Extension(3) der Datei darstellen.
Der Motor des Ziel-Laufwerks muss bei Aufruf eingeschalten sein.

b.) TEISK: Die RAM Variablen REG_AF1, REG_DE1, REG_HL1, REG_IX und REG16_8 bis REG16_8+12 müssen seit TEISI erhalten worden sein.

Aussprungbedingungen: Der Akku liefert Informationen über den Erfolg der Operation:
A = &FF => Die Datei wurde komplett auf Diskette geschrieben. Gut so!
A = &F0 => Die Datei wurde nur TEILWEISE auf Disk geschrieben, es muß nochmals nachgeladen werden, der Rest wird mit TEISK gesichert.
A = &00 => Es ist überhaupt kein DIR eingelesen.
A = &01 => Das gewählte LW ist nicht markiert.
A = &02 => Die Datei hat eine Länge von 0 KB. Also auch Abbruch.
A = &03 => Die Datei ist zu groß, auf Disk zu wenig Platz.
A = &04 => Im Ziel-DIR sind zu wenig Einträge frei.
A = &05 => Es sind keine KZS im E-RAM frei.
A = &06 => Bei schreiben meldet das LW "nicht bereit".

Bei TEISI & TEISK manipuliert: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY, RAM Variablen FDC_RES, REG08_0, REG08_1, REG_DE1(low), REG_PC(low).
Nur bei TEISI manipuliert: RAM-Variablen TURBO_A bis M, REG16_0 bis 9, REG32_1,2, REG_AF1, REG_HL1, REG_IX, das RAM von &B000 bis &BFFF.
Außerdem wurden die RAM-Konfiguration, die Spurwechselzeit der Laufwerke und die Inhaltsverzeichnisse verändert.

Beschreibung: Diese OS Funktion ist das Gegenstück zu TEILA. Wärend TEILA eine Datei teilweise läd, dienen TEISI / TEISK dazu die geladene Datei wieder abzuspeichern. Ganz oder in Stückchen.
Zuerst springt man nach TEISI. TEISI generiert einen Eintrag im Inhaltsverzeichniss und speichert alle KZS(Kurz-Zeit-Speicher) auf Diskette. Ein Fehler wird im Akku übergeben. Kehrt TEISI mit dem Byte &FF im Akku zurück, dann wurde die Datei komplett gesichert. Enthält der Akku jedoch den Wert &F0, so wurden zwar alle KZS gespeichert, aber die Datei ist länger als das zur Verfügung stehende RAM. Einige RAM-Variablen sind nun zu erhalten, siehe oben!! Es muß also erst der Dateirest mit TEILB nachgeladen werden, dann kann dieser Rest mit TEISK auf Disk geschrieben werden.

Dieser Vorgang kann sich mehrmals wiederholen, wenn die Datei lang oder das E-RAM gering ist. Das Inhaltsverzeichnis wird zwar mit den korrekten neuen Einträgen versorgt, jedoch wird es nicht auf Disk geschrieben. Dies hat z.B. dann Sinn, wenn mehrere Dateien kopiert werden sollen.

Bitte Beachten: Bei Aufruf der OS Funktion muß der Motor des Laufwerks an sein! Das neue DIR muß nach Aufruf von TEISI/K mittels SIDIR gesichert werden.
Vor Aufruf der OS Funktion kann es sinnvoll sein, einen weiteren Block (unterhalb des DIR Puffers) als DIR Puffer zu vermerken. Da neue Einträge generiert werden, steigt der ERAM Verbrauch im DIR Puffer.

EINE SEITE TEXT AUF DEM BILDSCHIRM DARSTELLEN

Kurzbeschreibung: Eine Seite Text wird auf dem Bildschirm dargestellt.

Label: TYSAZ (normal) // TYSZA (Anzahl RETURNs variabel)

ROM-Nummer: C

Startadresse: &FD6E (TYSAZ) // &FD6B (TYSZA)

Einsprungbedingungen: für alle Labels:
MAX_CRX = MODE 2 Spalten pro Zeile
MAX_CRY = Anzahl Zeilen pro Seite
REG08_6 = phys.RAM-Block (&C0,&C4,&C5,..,&FF), in dem der Text steht, dieser Block muß schon eingeblendet und als KZS markiert sein.
REG16_0 = Start der auszugebenden Seite Text
REG16_1 = Textende in momentaner RAM-Konfiguration, also Byte oberhalb des akt. KZS bzw. Byte oberhalb des Pufferendes.
Für TYSZA gilt außerdem:
XL = Anzahl bereits ausgegebener RETURNs, also Anzahl bereits verbrauchter Zeilen.

Aussprungbedingungen: gilt für alle Labels:
A = &00 => Seite konnte korrekt dargestellt werden.
A = &FE => Seite wurde ausgegeben, das Textende wurde erreicht!
A = &FF => Seite teilweise ausgegeben, bitte restlichen Text NACHLADEN! Dann TYSZA aufrufen und Rest ausgeben.

Manipuliert: AF, BC, DE, HL, AF', BC', DE', HL', REG08_6, REG16_0, 1, C_POS u.d. RAM-Status

Beschreibung: Diese OS Funktion dient dazu eine Seite Text auf dem Bildschirm darzustellen. Der Text kann irgendwo im RAM oder in den Kurzzeit-Speichern liegen. Als Einsprung benutzt man das Label TYSAZ.
Liegt der Text in den KZS, dann wird am Ende eines Blocks selbstständig der nächste KZS eingeblendet, und der dortige Text fertig ausgegeben.
Normalerweise wird die OS Funktion mit A = &00 zurückkehren. Dies bedeutet, daß die aktuelle Seite Text dargestellt wurde, aber der Text noch nicht zu Ende ist. Denn wenn das absolute Textende erreicht wurde, wird mit A = &FE beendet.
Kehrt die OS Funktion mit A = &FF zurück, dann ist das Ende aller KZS erreicht, das Dateiende aber noch nicht. Nun sind Daten z.B. mittels TEILB (siehe auch vorige OS Funktionen) nachzuladen. Anschließend kann der Rest der aktuellen Seite mittels TYSZA ausgegeben werden. Beim Einsprung in TYSZA muß XL mit der Anzahl bereits ausgegebener RETURNs geladen sein, dies erledigt normalerweise TYSAZ, also XL bitte intakt lassen (oder sichern).

Bitte Beachten: Bitte Ein- & Aus-Sprung-Bedingungen BEACHTEN! Beim Einsprung braucht TYSZA mehr Parameter als TYSAZ.

MULTIPLIKATION VON 8 MAL 8 BIT, MIT 16 BIT ERGEBNIS

Kurzbeschreibung: Zwei 8-Bit Werte werden miteinander multipliziert.

Label: MUL88

ROM-Nummer: C

Startadresse: &FD65

Einsprungbedingungen: Die Register H und L enthalten je einen 8 Bit Wert, diese sollen multipliziert werden.

Aussprungbedingungen: HL = 16 Bit Ergebniß der Multiplikation.

Manipuliert: DE, HL und die Flags

Beschreibung: Da der Z80 keine Multiplikationsbefehle hat, muß sich der Programmierer softwaremäßig behelfen.
Diese OS Funktion bietet eine sehr schnelle 8 * 8 Bit Multiplikation.
Dabei werden die beiden zu multiplizierenden Werte in den Registern H und L angegeben.
Die OS Funktion übergibt das 16 Bit Ergebniss im Doppelregister HL.
Eventuelle Vorzeichen werden nicht beachtet.

Bitte Beachten: Nur positive Werte im Bereich von 0 bis 255 multiplizierbar.

SICHERN EINER DATEI AUF DISKETTE

Kurzbeschreibung: Ein definierter Speicherinhalt wird als Datei auf Diskette oder HD gesichert.

Label: SICHRE

ROM-Nummer: C

Startadresse: &FD8C

Einsprungbedingungen:
Der Laufwerksmotor ist einzuschalten (OUT &FA7E,&FF).
Der SAVE-Modus muß in REG08_3 eingetragen werden (als ASCII-Zahl+1):
REG08_3 = "1"+1 = &32 => Vordergrund Programm sichern, Daten aus RAM.
REG08_3 = "2"+1 = &33 => Hintergrund Programm sichern, Daten aus RAM.
REG08_3 = "3"+1 = &34 => Hauptspeicher sichern.
REG08_3 = "4"+1 = &35 => Erweiterungs-RAM.
REG16_6+1 = Ziellaufwerk, auf das gesichert werden soll. Das Ziel-LW wird durch das entsprechende ASCII Zeichen symbolisiert, für Laufwerk A also z.B.: &41 (oder &61) => LW A (a) usw.

Bei SAVE-Modus 3 oder 4 sind weitere Daten anzugeben:
REG16_8 = 12 Bytes, geben die User-Nr. (1 Byte), Dateiname (8 Bytes) und Extension (3 Bytes) an.
REG_IX = Quell-Adresse, ab der gespeichert werden soll. (16 Bit).
AKT_RAM = Quell-Block, aus dem/ab dem gesichert werden soll. &C0-&FF.
REG_IY = Datei-Länge in KB (16 Bit). (Datei sollte max. 512K haben).

Aussprungbedingungen: Datei wurde auf Diskette / HD gespeichert und im IHV (Inhaltsverzeichnis) vermerkt.

Manipuliert: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY, und die RAM Variablen REG08_0,1,4, REG16_0 bis REG32_1, AKT_RAM, REG_R, der RAM-Bereich von &8000 bis &9FFF und &B000 bis &BFFF.
Außerdem wurden die E-RAM-Konfiguration, die Spurwechselzeit der Laufwerke und die Inhaltsverzeichnisse verändert. Ein DIR Puffer E-RAM ist eingeblendet, es sollte duch: LD BC,&7FC0:OUT (C),C wieder ausgeblendet werden.

Beschreibung: Dieses Unterprogramm erlaubt es einen beliebigen Speicherbereich als Datei auf Diskette oder HD zu schreiben und im Inhaltsverzeichniss zu vermerken.
Die Variable REG08_3 gibt an wie gesichert werden soll, es kann entweder ein bereits eingelesenes Programm gesichert werden, oder ein beliebiger RAM Bereich.
In jedem Fall ist in REG16_6+1 das Laufwerk von A..L anzugeben, und zwar als ASCII Wert von &41 .. &4D. Siehe Einsprungbedingungen!!!

Bitte Beachten: Es kann nur auf Laufwerke gespeichert werden, deren Inhaltsverzeichnis bereits eingelesen und in den Systemveariablen vermerkt ist.
Man sollte dafür sorgen daß sich auf das Laufwerk auch Daten schreiben lassen.
Das LW wird in REG16_6+1 als ASCII Wert angegeben.

ANZEIGE eines FutureOS DATEIHEADERS

Kurzbeschreibung: Ein FutureOS Dateiheader wird angezeigt.

Label: DHED

ROM-Nummer: C

Startadresse: &FD5F

Einsprungbedingungen: Es sollte eine markierte Datei vorhanden sein.

Aussprungbedingungen: Header Informationen wurden angezeigt!
XL = &8C => Datei hatte gar keinen Header !!!
XL = &B3 => Datei hatte Header (wie angenommen).
Das System befindet sich in MODE 1. Der 32 mal 32 Zeichen Modus ist aktiv. Das untere ROM (Zeichensatz!) ist eingeblendet.
Der angezeigte Header liegt ab &B400 im RAM.
Die bearbeitete Datei bleibt markiert, d.h. FTB erhalten!

Manipuliert: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY und die RAM-Variablen REG08_0,1,5, REG16_0, REG_PC und C_POS. Außerdem die RAM Bereiche von &B000 bis &B7FF und von &BC00 bis &BC7F, der RAM-Status, der Modus (1), der FDC, die Datei-Tagging Bytes und der Motor Status.

Beschreibung: Der 128 Byte Header einer Datei enthält unter FutureOS zusätzliche Informationen. (Kopfzeile, Icon - bitte Nachschlagen).
Diese OS Funktion dient dazu den Header der ersten markierten Datei auf dem Bildschirm anzuzeigen, dies beinhaltet folgende Informationen:
- Kopfzeile bzw. Semigrafik- bzw. Grafik-Icon
- Ladeadresse & Ladeblock
- Dateilänge
- Startadresse & Startblock

Zur Darstellung dieser Informationen wird MODE 1 benutzt, alle vier Farben finden Verwendung, sie sollten sich also etwas unterscheiden.

Bitte Beachten: Hat die Datei keinen Header bricht die OS Funktion ab und kehrt mit einer Fehlermeldung ins OS zurück.

LADEN DES DATEIHEADERS EINER MARKIERTEN DATEI

Kurzbeschreibung: Dateiheader der ersten markierten Datei laden.

Label: LADAH

ROM-Nummer: C

Startadresse: &FD92

Einsprungbedingungen: Es sollte eine markierte Datei vorhanden sein!

Aussprungbedingungen: 512 Bytes der ersten markierten Datei stehen ab &B400 im RAM. Der Header befindet sich also genau an dieser Adresse.
(REG_PC+1) = Code von 0 bis 14 des Mediums (Laufwerk, HD20, MM, SD-Karte) von dem die Datei geladen wurde

Manipuliert: siehe LADEN, dazu noch REG08_3, REG16_1

Beschreibung: Will man von einer Datei ihren ersten Sektor oder ihren Dateiheader laden, so kann man diese OS Funktion verwenden.

Es wird die erste markierte Datei verwendet. Sollte keine Datei markiert sein, dann bricht die OS Funktion mit einer Fehlermeldung ab, und springt ins OS zurück.

Nach Aufruf der OS Funktion steht der erste Sektor der Datei ab &B400 im RAM. Außerdem stehen User, Name und Extent der Datei ab &B7E0 im RAM.

Diese OS Funktion eignet sich besonders, wenn man den Header einer Datei untersuchen will. Die FTBs (Dateistatus) bleiben erhalten.

Bitte Beachten: Irgendeine Datei sollte markiert sein.

SICHERE L-RAM ALS LZS IM E-RAM

Kurzbeschreibung: Das RAM von &0000 bis &3FFF wird im Erweiterungs-RAM gesichert, dieses E-RAM wird als Lang-Zeit-Speicher (LZS) markiert. Siehe auch OS Funktion: RRB0.

Label: SSB0

ROM-Nummer: C

Startadresse: &FD4D

Einsprungbedingungen: Es sollte E-RAM vorhanden sein, davon sollte ein 16 KB Block als frei oder KZS markiert sein.

Aussprungbedingungen: Das Z-Flag gibt Aufschluß über den Erfolg:

- Zero-Flag geleert => es ist kein E-RAM frei, Unteres RAM nicht gesichert!
- Zero-Flag gesetzt => Unteres RAM wurde korrekt gesichert

Die RAM-Variable L_RAM enthält die Nummer des physikalischen E-RAM Blocks von &C4 bis &FF. Dieses Erweiterungs-RAM wurde eingeblendet.
Falls kein freies E-RAM gefunden wurde (Zero-Flag = 0) enthält die OS Variable L_RAM den Wert &00.
Das untere RAM wurde zwischen &0000 und &3FFF eingeblendet
Der Bildschirm Modus wurde auf 2 gesetzt

Manipuliert: AF, BC, DE, HL und die Variable L_RAM und XRAM_??.

Beschreibung: Unter FutureOS hat der unterste RAM Block, also das Lower-RAM (&0000 bis &3FFF) oft Pufferfunktion. Er wird z.B. von DIRWA und von TS_D_IN als Zwischenspeicher benutzt.
Die OS Funktion SSB0 sichert den Lower-RAM Block im Erweiterungs(E)-RAM, vorrausgesetzt dort sind noch 16 KB frei. Der freie E-RAM Block wird als Lang-Zeit-Speicher(LZS) markiert, außerdem wird die physikalische RAM-Konfiguration des E-RAMs in der Variable L_RAM gesichert.

Bitte Beachten: Ist Z-Flag gelöscht, so wurde das untere RAM (&0000-&3FFF) nicht gesichert.

HOLE ALTES L-RAM AUS LZS DES E-RAM

Kurzbeschreibung: Das RAM von &0000 bis &3FFF wird aus dem E-RAM restauriert. Siehe auch OS Funktion: SSB0. s.o.

Label: RRB0

ROM-Nummer: C

Startadresse: &FD4A

Einsprungbedingungen: SSB0 sollte das L-RAM zuvor gesichert haben

Aussprungbedingungen: Das Z-Flag gibt Aufschluß über den Erfolg.
Zero-Flag gesetzt => Fehler, L-RAM wurde nie gesichert!
Zero-Flag geleert => L-RAM wurde korrekt restauriert.
Der Zwischenspeicher-LZS des E-RAMs wurde wieder freigegeben.
Die Variable L_RAM wurde auf &00 gesetzt.

Manipuliert: AF, BC, DE, HL und die Variablen L_RAM und XRAM_??.

Beschreibung: Wurde das L-RAM (&0000 bis &3FFF) zuvor durch SSB0 im E-RAM gesichert, dann kann man es durch diese OS Funktion (RRB0) wieder restaurieren. Der als LZS markierte Block des E-RAMs wird wieder freigegeben und als KZS markiert. Außerdem wird die Variable L_RAM auf &00 gesetzt.
Ist beim Rücksprung das Z-Flag gelöscht, dann wurde das L-RAM korrekt restauriert. Bei gesetztem Z-Flag wurde das L-RAM gar nicht gesichert.

Bitte Beachten: L-RAM sollte zuvor (--> SSB0) gesichert worden sein.

DATEN DER RAM-DISK SCHÜTZEN - XRAM_?? ÄNDERN

Kurzbeschreibung: Die durch RAM-Disk-Dateien belegten Blöcke des E-RAMs werden geschützt.

Label: PRO_MD

ROM-Nummer: C

Startadresse: &FD50

Einsprungbedingungen: -

Aussprungbedingungen: Die von RAM-Disk-Dateien belegten Blöcke des E-RAMs werden in den XRAM_?? Variblen als USER-RAM (&11) vermerkt.
Die RAM-Konfiguration &CC (5. 16K Block) ist eingeblendet, dieser Block enthält das DIR der RAM-Disk.

Manipuliert: AF, BC, DE, HL, RAM-Status und die Variablen XRAM_CC..FF.

Beschreibung: Das FutureOS beinhaltet eine dynamische RAM-Verwaltung.
Es benutzt also das gesammte freie E-RAM. Unter Amsdos bzw. CP/M kann das E-RAM als RAM-Disk genutzt werden. Das Inhaltsverzeichniss (DIR) der RAM-Disk steht im Block &7FCC (5. 16K Block) des E-RAMs ab der Adresse &4000 im Speicher.
Diese OS Funktion schützt die Dateien der RAM-Disk. Dafür wird zunächst die höchste, von der RAM-Disk belegte, Blocknummer ermittelt.
Alle Blocks darunter werden in den XRAM_?? Variablen als USER-RAM (&11) markiert. Von diesen USER-RAMs laßt das FutureOS die Finger, somit werden alle Daten der RAM-Disk erhalten.
Ist die RAM-Disk komplett gefüllt, dann sind alle Variablen XRAM_CC bis XRAM_FF mit &11 markiert, dem FutureOS stehen dann nur noch die 64 KB der Blöcke &C4, &C5, &C6 und &C7 zur Verfügung.
Diese OS Funktion wird bei jedem System-Start aufgerufen, die Dateien der RAM-Disk werden also automatisch geschützt.
Hat man zuwenig RAM, dann kann man das DIR der RAM-Disk per Hand löschen, und das System neu starten. Das FutureOS wird sich nun wieder richtig wohl fühlen, aber man sollte nicht vergessen zuvor ein Backup der RAM-Disk anzufertigen.

Bitte Beachten: Auch Dateien die im DIR der RAM-Disk gelöscht wurden, also auf User &E5 stehen werden erhalten. Will man den "Müll" loswerden, dann kann man die Dateien entweder von Hand entfernen, indem man alle Blocknummern auf &00 setzt. Oder man kann alle RAM-Disk Dateien auf Disk kopieren, dann das DIR der RAM-Disk mit &E5 initialisieren und dann die Dateien zurückkopieren.

ERMITTLE DIE BLOCK-ANZAHL DER ERSTEN MARKIERTEN DATEI

Kurzbeschreibung: Die Blockanzahl der ersten markierten Datei wird bestimmt.

Label: G_BLK

ROM-Nummer: C

Startadresse: &FD41

Einsprungbedingungen: Es sollte mindestens eine Datei markiert sein.
Die System-RAM-Variablen sollten korrekte Werte enthalten.

Aussprungbedingungen: Wenn eine markierte Datei gefunden wurde, dann ist ihre Blockanzahl im Register BC gegeben. War keine Datei markiert, dann enthält das Register HL den Wert &0000.
REG_BC1 = BC = Anzahl Blöcke der ersten markierten Datei.
HL = &0000 => es war gar keine Datei markiert.

Manipuliert: siehe FMD32

Beschreibung: Will man die Block-Anzahl einer Datei bestimmen, dann kann man diese OS Funktion dazu benutzen. Es wird die Anzahl der Blöcke der ersten markierten Datei bestimmt.
Beim Einsprung ist darauf zu achten, daß das System-RAM korrekte Werte enthält, besonders die Variablen TURBO_?. Wenn sich der Programmierer an die Konventionen hält, dann ist das kein Problem.
Diese OS Funktion funktioniert für Data, System, IBM und Vortex Format, auch Dateien auf HD20 werden unterstützt.
Nach dem Aufruf der OS Funktion gibt diese im Register BC und in der RAM-Variable REG_BC1 die Block-Anzahl der ersten markierten Datei zurück.
Aber Achtung: Wenn das Register HL beim Aussprung den Wert &0000 enthält, dann wurde gar keine markierte Datei gefunden.
Man beachte: Eine Datei hat bei verschiedenen Formaten auch eine verschieden große Block-Anzahl:
Nehmen wir an, die Datei ‚MULTITAS.X16' ist 7 KB groß, dann hat sie unter Data, System und IBM Format auch die Blockanzahl von 7. Aber, unter Vortex-Format hat die selbe Datei nur zwei Blöcke und keine sieben.

Bitte Beachten: RAM-Variablen müssen korrekte Werte enthalten.

Ermittle die Blockanzahl einer Datei: Data, System, IBM oder Vortex Format

Kurzbeschreibung: Die Blockanzahl einer Datei wird bestimmt. Dabei muß es sich aber um Data, System, IBM oder Vortex Format handeln.

Label: BLK_FD

ROM-Nummer: C

Startadresse: &FD47

Einsprungbedingungen: HL = Zeiger auf ersten Eintrag der Datei.
Die System-RAM-Variablen sollten korrekte Werte enthalten.

Aussprungbedingungen: BC = &00?? = Anzahl Blöcke der Datei.

Manipuliert: AF, BC, DE, HL, IX und IY.

Beschreibung: Will man die Block-Anzahl einer Datei bestimmen, deren erster Extent im 32er DIR bekannt ist, dann kann man diese OS Funktion dazu benutzen. Die Datei muß sich aber auf einem Datenträger mit Data, System, IBM oder Vortex Format befinden.
Beim Einsprung ist darauf zu achten, daß das Register HL auf den ersten Eintrag der zu untersuchenden Datei zeigt. Damit ist der erste Eintrag im 32er DIR gemeint. Das entsprechende E-RAM, indem das DIR der Datei gepuffert ist, muß natürlich eingeblendet sein.
Nach dem Aufruf der OS Funktion gibt diese im Register BC die Block-Anzahl der Datei zurück.

Man beachte: Eine Datei hat bei verschiedenen Formaten auch eine verschieden große Block-Anzahl:
Nehmen wir an, die Datei BEER-TST.ASC ist 14K groß, dann hat sie unter Data, System und IBM Format auch die Blockanzahl von 14. Aber, unter Vortex-Format hat die selbe Datei nur vier Blöcke und keine 14.

Bitte Beachten: Die Datei muß sich auf einem Datenträger mit Data, System, IBM oder Vortex Format befinden.

ERMITTLE DIE BLOCK-ANZAHL EINER DATEI AUF FESTPLATTE HD20

Kurzbeschreibung: Die Blockanzahl einer Datei wird bestimmt. Dabei muß es sich aber um eine Datei auf Festplatte handeln.

Label: BLK_HD

ROM-Nummer: C

Startadresse: &FD44

Einsprungbedingungen: HL = Zeiger auf ersten Eintrag der Datei.
Die System-RAM-Variablen sollten korrekte Werte enthalten.

Aussprungbedingungen: BC = &???? = Anzahl Blöcke der Datei.

Manipuliert: AF, BC, DE, HL, IX und IY.

Beschreibung: Will man die Block-Anzahl einer Datei bestimmen, deren erster Extent im 32er DIR bekannt ist, dann kann man diese OS Funktion dazu benutzen. Die Datei muß sich aber auf der Dobbertin HD befinden.
Beim Einsprung ist darauf zu achten, daß das Register HL auf den ersten Eintrag der zu untersuchenden Datei zeigt. Damit ist der erste Eintrag im 32er DIR gemeint. Das entsprechende E-RAM, indem das DIR der Datei gepuffert ist, muß natürlich eingeblendet sein.
Nach dem Aufruf der OS Funktion gibt diese im Register BC die Block-Anzahl der Datei zurück.
Achtung: Würde sich die Datei auf Diskette befinden, dann könnte sie eine erheblich abweichende Block-Anzahl aufweisen.

Bitte Beachten: Die Datei muß sich auf HD befinden.

WARTEN AUF DEN DRUCK EINER BELIEBIGEN TASTE

Kurzbeschreibung: OS Funktion wartet auf den Druck irgendeiner Taste.

Label: WATA

ROM-Nummer: C

Startadresse: &FD38

Einsprungbedingungen: -

Aussprungbedingungen: Vom Anwender wurde ein Taste gedrückt. Der Wert dieser Taste + 1 wird in Register A übergeben (siehe H_ALLET).

Manipuliert: AF, BC, DE, HL, IX, PIO und PSG

Beschreibung: Diese OS Funktion dient dazu so lange zu warten, bis durch den Anwender eine beliebige Taste gedrückt wird. Die OS Funktion übergibt den ASCII - Wert der "gedrückten Taste + 1" im Akku. Ohne Tastendruck wartet die Funktion für ewig.
Um die Tastatur abzufragen sollte allerdings die OS-Funktion ‚H_XALLET' verwendet werden.

Bitte Beachten: OS Funktion kehrt erst NACH dem Druck einer Taste zurück.

KONVERTIERE ZWEI ASCII-ZEICHEN IN EINEN 8 BIT WERT

Kurzbeschreibung: Zwei ASCII-Zeichen werden in einen 8 Bit Wert konvertiert.

Label: CC2N

ROM-Nummer: C

Startadresse: &FD3B

Einsprungbedingungen: Ab HL liegen zwei ASCII-Zeichen (0-9, A-F) im RAM. Zuerst das höherwertige, anschließend das niederwertige Zeichen. Ein Beispiel ist die Eingabe einer User Nummer in einem Text String.

Aussprungbedingungen: A = 8 Bit Wert &00-&FF
HL = HL + &0001

Manipuliert: AF, B und HL

Beschreibung: Diese OS Funktion dient dazu zwei ASCII-Zeichen, die im Speicher abgelegt sind in einen 8 Bit Wert zu konvertieren. Bei Aufruf der OS Funktion zeigt HL auf zwei Zeichen im Speicher: (HL) enthält das höherwertige Zeichen und (HL+1) enthält das niederwertig Zeichen.

Nach dem Rücksprung der OS Funktion ist Register HL um Eins erhöht und der Akku enthält einen 8 Bit Wert, der sich folgendermaßen errechnet:

A = (HL)*16 + (HL+1)

Beispiel:

Die Adresse **&4000** enthält das Zeichen **"7"** und **&4001** enthält **"B"**.

```
LD   HL,&4000
CALL CC2N
```

Der Akku enthält nun &7B.

Diese OS Funktion eignet sich lediglch dazu Zeichen von "0" bis "9" und "A" bis "F" in einen 8 Bit Wert zu konvertieren.

Im FutureOS wird diese OS Funktion z.B. dazu benutzt, mit der Hand eingegebene User-Nummern (2 Zeichen) in ein Byte umzurechnen.

Bitte Beachten: Die zu konvertierenden Zeichen müssen zwischn "0" und "9" bzw. "A" und "F" sein.

Ein BYTE wird in Form ZWEIER Zeichen auf dem BILDSCHIRM dargestellt

Kurzbeschreibung: Ein Byte wird hexadezimal dargestellt.

Label: HAUT

ROM-Nummer: C

Startadresse: &FD3E

Einsprungbedingungen: A = Byte, das dargestellt werden soll.

Aussprungbedingungen: Zwei hexadezimale Zeichen wurden auf den Bildschirm geschrieben.

Manipuliert: AF, BC, DE, HL, AF', IX, CUR_POS und REG08_0.

Beschreibung: Diese OS Funktion dient dazu einen 8 Bit Wert hexadezimal auf dem Bildschirm darzustellen. Das darzustellende Byte wird im Akku übergeben. Die OS Funktion gibt anschließend zwei hexadezimal Zeichen auf dem Bildschirm aus. Dabei muss Bildschirm Modus 1 aktiviert sein. Die beiden Zeichen werden mit PEN 1 (BB!) ab der aktuellen CURsor-POSition dargestellt.

Bitte Beachten: OS Funktion ist für Mode 1 ausgelegt. Das Byte wird im hexadezimalen Zahlensystem dargestellt.

Teste ob 4 MB RAM Erweiterung angeschlossen, setze SYSTEM Variablen

Kurzbeschreibung: Testet wie viel Erweiterungs - RAM (0,5 MB bis 4 MB) am CPC angeschlossen ist. Die ersten 512 KB werden nicht beachtet.

Label: TXR4M

ROM-Nummer: C

Startadresse: &FD32

Einsprungbedingungen: -

Aussprungbedingungen: System-Variablen X4RXE, X4RDC, X4RBA und X4R98 wurden entsprechend des angeschlossenen Erweiterungs-RAMs gesetzt.
Das Register BC enthält &7FC0 und der Hauptspeicher ist eingeblendet.

Manipuliert: AF, BC, DE, HL, AF', IX, Adresse &4000 in allen 16 KB Erweiterungs-RAM Blöcken ab der RAM Konfiguration &7EC4. Und die RAM Konfiguration selbst: Hauptspeicher ist eingeblendet.
Die ersten 512 KB E-RAM wurden jedoch nicht manipuliert.

Beschreibung: An den CPC kann eine 4 MB RAM-Erweiterung angeschlossen werden. Dieses Erweiterungs-RAM wird in 16 KB Blöcken zwischen &7FC4 und &78FF angesprochen.

Diese OS Funktion testet welche einzelnen 512 KB Blöcke der möglichen 4 MB RAM tatsächlich angeschlossen / ansprechbar sind. Die ersten 512 KB (Port &7Fxx) werden dabei nicht beachtet.

Für jeden angeschlossenen 512 KB Block werden die entsprechenden Bits in den System RAM Variablen X4RXE, X4RDC, X4RBA bzw. X4R98 gesetzt (siehe Datei #OS-VAR.DEU). Diese 512 KB Blöcke werden jeweils mittels der I/O-Ports &7Exx, &7Dxx, &7Cxx, &7Bxx, &7Axx, &79xx bzw. &78xx angesprochen. Die ersten 512 KB (Port &7Fxx) werden nicht beachtet, da diese über die XRAM_?? Variablen verwaltet werden.

Nach dem Rücksprung der OS Funktion TXR4M sind die System RAM Variablen zur Verwaltung von 4 MB E-RAM auf gültige Werte gesetzt.

(Das FutureOS ruft TXR4M beim Start des OS einmal auf). Weiterhin wurde die Adresse &4000 eines jeden 16 KB Blocks oberhalb des ersten 512 KB Blocks manipuliert, dies entspricht den RAM Konfigurationen &7Exx, &7Dxx, &7Cxx, &7Bxx, &7Axx, &79xx und &78xx.

Bitte Beachten: Die ersten 512 KB Erweiterungs-RAM (&7Fxx) werden nicht beachtet und nicht verändert.
Das Erweiterungs-RAM, das ab Port &7Exx bis Port &78xx angesprochen wird, wurde in jedem 16 KB Block an Adresse &4000 manipuliert.

Physikalische E-RAM Auswahl in Zeiger auf XRAM-Variable umrechnen

Kurzbeschreibung: Konvertiert physikalische E-RAM Auswahl (der ersten 512 KB Erweiterungs-RAM) in einen Zeiger auf eine der Variablen XRAM_C4..._FF.

Label: E2XRAM

ROM-Nummer: C

Startadresse: &C9D4

Einsprungbedingungen: A = physikalische E-RAM Auswahl &C4...&FF

Aussprungbedingungen: HL = Zeiger auf eine der System-Variablen XRAM_C4 bis XRAM_FF

Manipuliert: AF und HL

Beschreibung: An den CPC können bis zum 4 MB Erweiterungs RAM (E-RAM) angeschlossen werden. Dabei kommen den ersten 512 KB eine besondere Rolle zu. Sie werden über den Port &7Fxx angesprochen. Um ein E-RAM zwischen &4000 und &7FFF einzublenden wird einer aus 32 Werten an den Port &7Fxx gesendet (&C4, &C5, &C6, &C7, &CC, &CD, ... &FF). Dieser Wert wird als physikalische E-RAM Auswahl bezeichnet. Für jeden dieser 32 E-RAM Blöcke steht im FutureOS eine XRAM Variable zur Verfügung, die Aufschluss über die Verwendung dieses 16 KB Blocks gibt. Details dazu sind im Handbuch und in Datei ‚**#OS-VAR.DEU**‘ zu finden.

Um nun den Wert einer solchen physikalischen E-RAM Auswahl in einen Zeiger auf die entsprechende XRAM Variable umzurechnen kann die OS Funktion E2XRAM verwendet werden.

Dabei wird die physikalische E-RAM Auswahl in A übergeben. Nach der Rückkehr der OS Funktion zeigt HL auf die zugehörige XRAM Variable.

Bitte Beachten: Diese OS Funktion bedient nur die ersten 512 KB E-RAM (&7Fxx), denn nur für diese 32 Blöcke von je 16 KB E-RAM stehen XRAM Variablen zur Verfügung.

XRAM-VARIABLE IN PHYSIKALISCHE E-RAM AUSWAHL UMRECHNEN

Kurzbeschreibung: Ein Zeiger auf eine der 32 XRAM-Variablen wird in die zugehörige physikalische Erweiterungs-RAM Auswahl umgerechnet.

Label: BJKG

ROM-Nummer: C

Startadresse: &C9F9

Einsprungbedingungen: HL = Zeiger auf OS Variable XRAM_C4...FF

Aussprungbedingungen: Der Akku A enthält die physikalische Nummer des zugehörigen 16 KB E-RAM Blocks, also die Hardware E-RAM Auswahl von &C4 bis &FF.
Und Register B wurde mit dem Wert &7F geladen um einen 'OUT (C),A' zu ermöglichen.

Manipuliert: AF und B

Beschreibung: Für die Verwaltung der ersten 512 KB Erweiterungs-RAM (E-RAM) stellt das FutureOS die XRAM-Variablen zur Verfügung. Sie tragen die Bezeichnungen XRAM_C4, XRAM_C5 ... XRAM_FF. Jede der 32 Variablen gibt Aufschlu: über die Verwendung des zugehörigen 16 KB großen E-RAM Blocks. Diese Blöcke werden physikalisch über Port &7Fxx mittels der Bytes &C4, &C5 ... &FF ausgewählt. Ein Beispiel:

```
LD  BC,&7FC4 ;Auswahl des ersten E-RAMs (&C4) über den Port &7Fxx
OUT (C),C    ;Das E-RAM im Bereich von &4000 bis &7FFF einblenden
```

Die OS Funktion BJKG dient nun dazu eine der XRAM Variablen in die entsprechende Hardware E-RAM Auswahl umzurechnen. Dabei muss HL beim Einsprung auf eine der XRAM-Variablen zeigen. Nach dem Rücksprung enthält der Akku den Wert &C4 - &FF für die Hardware E-RAM Auswahl.
Und Register B enthält den Wert &7F. Mit dem Kommando 'OUT (C),A' kann man den berechneten E-RAM Block direkt einblenden.

Bitte Beachten: Nach dem Rücksprung dieser OS Funktion ist es möglich das berechnete E-RAM direkt einzublenden:

```
OUT (C),A ;E-RAM einblenden (zwischen &4000 und &7FFF)
```

FREIGEBEN EINES 16 KB E-RAM BLOCKS DER ERSTEN 512 KB E-RAM

Kurzbeschreibung: 16 KB Block des E-RAMs (erste 512 KB) freigeben.

Label: FER7F

ROM-Nummer: C

Startadresse: &C9CB

Einsprungbedingungen: A = physikalische E-RAM Auswahl &C4...&FF

Aussprungbedingungen: HL = Zeiger zugehörige Sys.-Variable XRAM_C4-FF

Manipuliert: AF, HL und die entsprechende XRAM_?? Variable

Beschreibung: Die OS Funktion FER7F dient zur Freigabe eines zuvor belegten Blocks des Erweiterunngs-RAMs. Jeder dieser Blöcke ist 16 KB groß und jeweils 512 KB (also 32 Blöcke) werden über einen Port angesprochen.

OS Funktion FER7F bezieht sich auf die ersten 512 KB (Port &7Fxx) der maximal möglichen 4 MB E-RAM des CPC.

Im FutureOS kommen den ersten 512 KB eine besondere Rolle zu, für jeden 16 KB Block existiert eine eigene 8 Bit System-Variable. Es handelt sich dabei um XRAM_C4, _C5, bis _FF. Es gibt 32 von ihnen.

Diese XRAM Variablen geben über Vorhandensein und die Nutzung des entsprechenden E-RAMs Aufschluß. Physikalisch werden diese 16 KB E-RAM Blöcke über den Port &7Fxx angesprochen. Um ein E-RAM zwischen &4000 und &7FFF einzublenden wird einer aus 32 Werten an den Port &7Fxx gesendet (&C4, &C5, &C6, &C7, &CC, &CD, ... &FF). Dieser Wert wird als physikalische E-RAM Auswahl bezeichnet. Details dazu sind im Handbuch und in Datei #OS-VAR.DEU zu finden.

Um nun ein zuvor benutztes E-RAM wieder freizugeben läd man die physikalische E-RAM Auswahl in Register A und ruft anschließend die OS Funktion FER7F auf. Sie gibt den E-RAM Block durch Anpassung seiner XRAM Variable frei. Nach der Rückkehr der OS Funktion zeigt HL auf die zugehörige XRAM Variable.

Bitte Beachten: Diese OS Funktion bedient nur die ersten 512 KB E-RAM (&7Fxx), denn nur für diese 32 Blöcke von je 16 KB E-RAM stehen XRAM Variablen zur Verfügung.

SUCHE FREIES E-RAM in XRAM Variablen -> KONVERTIERE in PHYS. E-RAM

Kurzbeschreibung: In den 32 XRAM_C4..FF Variablen wird ein freies bzw. als Kurzzeitspeicher (KZS) markiertes E-RAM gesucht. Anschließend wird daraus der physikalische E-RAM Wert ermittelt (nur erste 512 KB).

Label: KZS2E

ROM-Nummer: C

Startadresse: &C9EA

Einsprungbedingungen: Die Variablen XRAMC4-FF enthalten korrekte Werte und es ist mindestens ein 16 KB Block frei/KZS.

Aussprungbedingungen:
A = Physikalische E-RAM Nummer &C4...&FF
B = &7F (ermöglicht direkten OUT (C),A Befehl)
HL = Zeiger auf zugehörige XRAM_C4..FF Variable

Manipuliert: AF, B und HL

Beschreibung: OS Funktion KZS2E bedient die ersten 512 KB (Port &7Fxx) der möglichen 4 MB E-RAM des CPC. Für jeden 16 KB Block der ersten 512 KB E-RAM existiert eine eigene 8 Bit System-Variable. Es handelt sich dabei um die 32 Variablen XRAM_C4, C5, C6, C7, CC, CD...FF. Diese XRAM Variablen geben über Vorhandensein und die Nutzung des entsprechenden E-RAMs Aufschluß. Physikalisch werden diese 16 KB E-RAM Blöcke über den Port &7Fxx angesprochen. Um ein E-RAM zwischen &4000 und &7FFF einzublenden wird einer aus 32 Werten an den Port &7Fxx gesendet (&C4, &C5, &C6, &C7, &CC, &CD, ... &FF). Dieser Wert wird als physikalische E-RAM Auswahl bezeichnet. Details dazu sind im Handbuch und in Datei #OS-VAR.DEU zu finden.
Diese OS Funktion sucht einen freien bzw. als Kurzzeit-Speicher markierten 16 KB E-RAM Block in den XRAM Variablen. Nach der Rückkehr der OS Funktion zeigt HL auf die zugehörige XRAM Variable. Ausserdem wird daraus die physikalische E-RAM Auswahl berechnet (Register A). Um den Befehl 'OUT (C),A' zu ermölgichen wird Register B mit dem Wert &7F geladen. Beispiel:

```
CALL KZS2E    ;E-RAM ermitteln
OUT  (C),A    ;und zwischen &4000 und &7FFF einblenden
```

Da diese OS Funktion nicht gesondert prüft, ob überhaupt ein E-RAM frei ist sollte sich die Applikation zuvor davon überzeugt haben (siehe OS Funktionsn zur E-RAM Verwaltung im Handbuch). Denn ist kein E-RAM frei / KZS so wird ein inkorrekter Wert übergeben.

Bitte Beachten: Es MUSS zumindest ein freier E-RAM Block vorhanden sein, bzw. mindestens ein Block als KZS markiert sein.
Diese OS Funktion arbeitet mit den ersten 512 KB E-RAM (&7Fxx), denn nur für diese 32 Blöcke von je 16 KB E-RAM stehen XRAM Variablen zur Verfügung.

LADEN UND DARSTELLEN EINES BILDES (CPC 17 KB, OCP KOMPRIMIERT)

Kurzbeschreibung: Ein normales CPC Bild (17 KB) oder ein OCP Bild werden geladen und dargestellt. Bildschirm-Format und MODE lassen sich dynamisch einstellen (Joystick bzw. Cursor-Tasten).

Label: ZEIBI bzw. ZEIBJ

ROM-Nummer: C

Startadresse: &FD2F (ZEIBI) bzw. &FD2C (ZEIBJ)

Einsprungbedingungen: Für beide Einsprünge gilt, dass eine Bild-Datei in the File-Tagging-Bytes(FTB) markiert sein muss. Die erste markierte Datei wird benutzt.
Bei ZEIBI muss der Datei-Header dieser Datei an &B400 geladen sein
Bei ZEIBJ enthält H das High-Byte (HH) eines Zeigers auf den Header einer Datei. Das low-Byte dieser Adresse ist immer &00: Adr. = &HH00

Aussprungbedingungen: Das Zero Flag gibt Aufschluss über den Erfolg der Aktion. Wenn das Zero-Flag gesetzt ist (Z=1) wurde das Bild erfolgreich dargestellt.
Sollte das Zero-Flag jedoch gelöscht sein (Z = 0), dann hat es sich bei der Datei entweder nicht um ein Bild gehandelt, oder es trat ein anderer Fehler auf.

Manipuliert: AF, BC, DE, HL, AF', BC', DE', HL', IX, IY und die RAM-Variablen REG08_0,1,5, REG16_0, REG_PC, L_RAM und XRAM_??. Außerdem die RAM Bereiche von &B000 bis &B7FF und von &BC00 bis &BC7F, der FDC, die PIO, der PSG, die Datei-Tagging Bytes, das Video-RAM (&C000-&FFFF) und der Motor Status.

Beschreibung: Die OS Funktion ZEIBI bzw. ZEIBJ dienen dazu ein Bild zu landen und darzustellen. Dabei kann man mittels Joystick bzw. Cursor-Tasten den Bildschirm-Modus (rechts - links) bzw. das Bildschirm-Format (auf - ab) verändern.
Vor dem Aufruf dieser OS Funktionen muss entweder ein Datei-Header an &B400 geladen werden (ZEIBI) oder das High-Byte eines Datei-Headers in Register H geladen werden (ZEIBJ). Beim Laden des Datei-Headers sollte A mit &FF geladen sein, um die FTBs zu erhalten (siehe OS Funktion LADEN). Der Datei-Header muss dabei an den Anfang einer Page (&XX00) geladen werden. Die vermutliche Bild-Datei muss weiterhin in den File-Tagging-Bytes (FTB, siehe Handbuch) als erste markiert sein.
Beim Aufruf einer der beiden OS Funktionen wird zuerst ermittelt, ob es sich tatsächlich um ein Bild handelt. Wenn ja, dann wird das Bild geladen, nötigenfalls dekomprimiert (OCP Advanced Art Studio Bild) und anschließend dargestellt. Während er Darstellung kann der Bildschirm-MODE geändert werden (rechts, links). Und es kann zwischen 80 x 25 bzw. 64 x 32 Zeichen (MODE 2 Referenz) gewechselt werden (auf, ab).
Dabei können sowohl Joystick als auch Curstor-Tasten benutzt werden. Der Druck auf Feuer 1 bzw. Copy beendet die Darstellung des Bildes.
Beide OS Funktionen setzen das Zero-Flag, wenn alles funktioniert hat. Ist das Zero-Flag jedoch gelöscht so trat ein Fehler auf (z.B. die Datei ist gar kein Bild).

Bitte Beachten: Es muss sich ein Header ab &B400 befinden und die zugehörige Datei muss markiert (getagged) sein. Ansonsten kommt es zu einer Fehlermeldung (Zero Flag gelöscht).

GENERIERE TD... VARIABLEN FÜR SCRI ETC.

Kurzbeschreibung: RAM-Variablen DIRIN, TDRAM, TDHST, TDANZ, TDAKT und TDLWK werden für OS Funktionen wie z.B. SCRI neu gesetzt.

Label: GEN_TDV

ROM-Nummer: C

Startadresse: &FD29

Einsprungbedingungen: RAM-Variablen TURBO_A...M korrekt gesetzt

Aussprungbedingungen: RAM-Variablen DIRIN, TDRAM, TDHST, TDANZ, TDAKT und TDLWK neu gesetzt

Manipuliert: AF, BC, DE, HL und die RAM-Variablen DIRIN, TDRAM, TDHST, TDANZ, TDAKT und TDLWK

Beschreibung: OS Funktionen wie SCRI, SC_AU oder SC_AB benötigen die aktuellen Werte in den System-Variablen DIRIN, TDRAM, TDHST, TDANZ, TDAKT und TDLWK.

Diese OS Funktion GEN_TDV setzt diese Variablen neu.

Dazu werden die Variablen TURBO_A, _B, _C, _D bis TURBO_M ausgelesen, diese müssen dementsprechend korrekte Werte enthalten.

Nach dem Löschen mehrerer Dateien bzw. dem Formatieren einer Diskette sollte diese OS Funktion (GEN_TDV) aufgerufen werden. Das wird durch das Desktop erledigt.

Für den Anwender bzw. eine Applikation ist GEN_TDV nur zu benutzen, wenn man direkt mit Inhaltsverzeichnissen arbeitet. In solchen Fällen wird GEN_TDV nach dem Schreiben von Inhaltsverzeichnissen aufgerufen.

Bitte Beachten: Beim Einsprung müssen die RAM-Variablen TURBO_A bis TURBO_M korrekt gesetzt sein.