Datei: FutureOS.DOK.docx (78 Seiten) Stand: 7. Januar 2024

Das FutureOS – Handbuch, Philosophie und Konzeption

I. Die Philosophie des Future Operating System

II. Die Konzeption des FutureOS

III. Der Aufbau des Operating System

IV. Das Desktop

V. Der Maschinen-Monitor

VI. Genauere Erklärung der einzelnen Fähigkeiten des OS

VII. Anhänge

Bitte lesen Sie sich diese Anleitung in aller Ruhe durch, so vermeiden Sie unter Umständen einige Fehler.

I. Die Philosophie des Future Operating System

Der Ursprung des Future Operating Systems

Die Idee zum FutureOS entstand Ende der 1980er Jahre. Ein Ziel war es, ein Betriebssystem (BS) zu erschaffen, daß sämtliche Erweiterungen des CPC bedienen kann. Vorhandene Erweiterungen sind oftmals nicht gleichzeitig verwendbar. Als Beispiel ist nur die Inkompatibilität diverser Disketten Betriebssysteme (AmsDOS, V-DOS, X-DDOS) zu nennen.

Außerdem ging es von vorneherein auch darum extrem schnelle Low-Level Routinen zu entwickeln. Die Zeichenausgabe war auch in den 80er Jahren schon quälend langsam.

Später kam zu diesem Konzept eine grafische Benutzeroberfläche – Turbo Desktop - hinzu, deren Funktion es ist Standart-Aufgaben schnell und effizient erledigen zu können.

Weiterhin bietet FutureOS einige Neuerungen, die weder auf dem CPC bzw. anderen Rechnern vorhanden sind: Beispielsweise die gleichzeitige Arbeit mit mehreren Laufwerken und HD-Partitionen, Programme die am Stück bis zu vier Megabyte erreichen dürfen, multidirektionales Kopieren von Dateien oder das I/O Porting-System ...

FutureOS ist bewußt als Ultrabetriebsystem konzipiert, das speziell auf den CPC zugeschnitten ist.

Die Geschwindigkeit, oder darfs ein bißchen schneller sein?

Aufgrund der Philosophie des FutureOS wurde größter Wert auf maximale Geschwindigkeit gelegt. Der Verbrauch an Speicher ist zweitrangig. Es war von vorneherein geplant mit 64 KB (Pseudo-EP)ROM zu arbeiten. Zusätzlich läßt sich das OS mit ROMs erweitern, z.B. dem IDE ROM.

Der CPC hat eine Z80 mit 4 MHz. Um in der heutigen Zeit mit moderneren Rechnern konkurrieren zu können, müssen die "Schwächen" der Hardware, vor allem aber der Software, eben durch die neue Software des FutureOS ausgeglichen werden.

Nebenbei, durch das gezielte Optimieren eines Programms, ist es möglich es um zwei Größenordnungen zu beschleunigen.

Beispiel: In der Demo "ZeichenDemo" auf der OS Disk werden eine halbe Million Zeichen in 23 Sekunden ausgegeben. Die 32 Bit CPU des Acorn A5000 braucht dafür immerhin noch 20 Sekunden.

Der Preis dieser Geschwindigkeit ist der Verzicht auf Kompatibilität zu den alten Programmen. Trozdem ist die Datei-Struktur von FutureOS 100% abwärtskompatibel, zusätzlich wurde sie stark erweitert. FutureOS ist das ideale Betreibssystem für Programmierer, die ihren CPC vollkommen ausreizen wollen.

Die Hardware und seine Verwaltung im OS

Jegliche Pheripherie des CPC wird durch ihren eigenen Treiber bedient. Dementsprechend sind die Low-Level-Funktionen sehr effizient.

FutureOS vermeidet jegeliche Art von Abstraktionsebene, die ein OS bekanntermaßen nur langsam machen würde.

Dem Programmierer wird im folgenden auch die direkte Programmierung der Hardware (durch kurze Listings) nähergebracht, um Applikationen noch effizienter gestalten zu können.

FutureOS stellt komfortable OS Funktionen zur Speicher-Verwaltung von bis zu 4 MB bereit. Programme dürfen eine entsprechende Länge haben.

Der Preis, oder warum kostet das so wenig

FutureOS ist kein kommerzielles Projekt. Sein Sinn besteht darin, den CPCs und ihren Usern neue Möglichkeiten zu eröffnen.

Warum sollte sich jemand ständig neue Hardware kaufen, wenn es genügt die Software effizient zu gestalten. Wir nutzen unsere CPCs ☺

Die persönliche User Idenfifikations Nummer

Ein Vorteil der personalisierten Version ist, neben der persönlichen Anpassung des OS, die persönliche Identifikationsnummer(ID) des Users. Diese User ID erlaubt es sich in Netzwerken anzumelden und persönliche Nachrichten zu erhalten. Eine Reihe von Applikationen benutzt die User ID um sich spezifisch an den Anwender anzupassen. Auch das Disketten-Magazin FutureView liest die User ID und blendet beim Laden ein Bild des Users ein und gibt eine persönliche Nachricht an den User aus.

Die Mißbrauchsgefahr der User ID ist gering, da sie in den FutureOS ROMs verankert ist.

II. Die Konzeption des FutureOS

Allgemein

Das FutureOS besteht - logisch gesehen - aus drei horizontalen Ebenen. Ganz unten existieren die Low-Level Funktionen, mit ihnen wird die Hardware direkt bedient. Auf mittlerer Ebene befinden sich die High-Level Funktionen, sie bedienen sich der Low-Level Funktionen um komplexere Funktionen auszuführen. Das Desktop steht über allem, es befehligt das System und sorgt für den Kontakt zum Anwender.

Logisch läßt sich das OS zwar in drei Ebenen gliedern, in der Realität ist es allerdings nach pragmatischen Gesichtspunkten aufgebaut. Aus Gründen maximaler Effizienz = Geschwindigkeit muß das drei Ebenen System oft verlassen werden.

Low-Level OS Funktionen

Sie decken alle grundlegenden Bereiche ab, wie Tastaturverwaltung, Ausgabe von Zeichen und Strings auf Bildschirm oder Drucker, Speicher-Management, Speicher kopieren oder füllen, Disketten- & Festplatten-Verwaltung, externe Hardware, sowie die Darstellung von Icons etc.

High-Level OS Funktionen

Die High-Level OS Funktionen sind sehr vielfältig. Einige sind noch nah an der Hardware, andere dagegen sehr hochentwickelt und neuartig. Z.B.: Speicherdump, dynamisches Erweiterungs-RAM (=E-RAM) Management bis 4 MB, Ladetabellen, Schreiben & Lesen & Format von Disks bzw. Harddisc, externe Hardware, Verwaltung der Inhaltsverzeichnisse usw.

Kontrolle durch das Desktop

Das Turbo-Desktop ist die Schnittstelle zwischen biologischer und künstlicher Intelligenz (Mensch und CPC). Seine Funktionen sind sehr vielfältig:

- Starten von Programmen
- Laden von Dateien
- Texte & Bilder anschauen
- Umbenennen von Dateien
- Kopieren von Disketten
- Formatieren von Disketten
- Aufruf des Maschinen-Monitors
- Drucken von Dateien und Inhalts-Vz.
- Speichern von Dateien
- Anschauen von Datei-Headern
- Löschen von Dateien
- Kopieren von Dateien
- Einstellen von Zeit & Datum, Alarm
- und vieles mehr ...

Abgesehen davon bietet das Desktop auch die Möglichkeit ein Programm direkt zu unterstützen. So kann ein Programm das Desktop ähnlich einer Unterroutine aufrufen, um mit Hilfe des Turbo-Desk Funktionen auszuführen. Anschließend führt das OK Icon den Anwender ins Programm zurück. Dadurch werden Programme ähnlicher und leichter bedienbar.

Gliederung in Desktop, Monitor und Umgebung (API)

Das OS läßt sich auch vertikal dreiteilen. Das Desktop, der Monitor und die Umgebung, also das Applikations-Programmierer-Interface (API) bilden die drei Säulen des OS. Während das Desktop für die Verwaltung von Daten und Dateien zuständig ist, bietet der Monitor dem Anwender die Möglichkeit direkt auf die Hardware zuzugreifen.

Neben Desktop und dem Monitor stellt die Umgebung (API) die dritte und wichtigste Säule des OS dar. Sie ermöglicht es dem Programmierer sehr effiziente Programme zu generieren. Eine OS Bibliothek hilft dabei.

III. Der Aufbau des Operating System

Speicherverwaltung

Zuerst einige Informationen zu den ersten 64 KB und den enthaltenen System-Variablen. Anschließend noch einige Worte zum Erweiterungs-RAM. Bitte sehen Sie sich dazu auch die Dateien **#EQU-API.DEU** und **#OS-VAR.DEU** an.

Das FutureOS ist auf vier 16 KB ROMs aufgeteilt, die mit A, B, C und D bezeichnet werden. Ihre physikalischen ROM Nummern können allerdings beliebig sein. Lediglich ROM A muss eine Nummer zwischen 1 und 15 haben. Die ROM-Nummern der OS ROMs sind fix, können aber durch ein Programm angepasst werden. Nur jeweils ein ROM kann zwischen &C000 und &FFFF eingeblendet werden.

Das Erweiterungs-RAM wird in 16 KB Blöcken verwaltet und von &4000-&7FFF eingeblendet.

Memory Map des Future Operating System

Das Standart-RAM (die ersten 64 KB) läßt sich in vier 16 KB Blöcke (0, 1, 2 und 3) einteilen.

Block 0: Von &0000 bis &3FFF
Lesen: 16K RAM oder L-ROM, je mit Zeichensatz von &3800 bis &3FFF
Schreiben: 16K RAM (RST Vektoren bei Interrupt-Modus 1)

Block 0 wird vom Desktop als Puffer verwendet wird. Programme können diesen Block freizügig nutzen. Den RAM-Zeichensatz (&3800-&3FFF) bitte erhalten (falls geladen).

Block 1: Von &4000 bis &7FFF
Lesen: 16K RAM / Standart RAM oder Erweiterungs-RAM (0-31 - 255),
Schreiben: 16K RAM / beziehungsweise Memory Mapped I/O (CPC Plus)

Normalerweise ist das Standart-RAM eingebledet (Konfig &7FC0), es kann aber auch einer aus 32 bzw. 255 Erweiterungs-RAM Blöcken (je 16K) aktiv sein. (Konfiguration: &C4,&C5, ... ,&FF / 4 MB: Highb. &78XX- &7FXX).

Block 2: Von &8000 bis &BFFF
Lesen: 16K RAM einziger COMMON Block
Schreiben: 16K RAM einziger COMMON Block

Dieser 16 KB Block ist unter FutureOS immer eingeblendet, er enthält die Systemvariablen. Das OS belegt maximal 8 KB (&A000 bis &BFFF).

Block 3: Von &C000 bis &FFFF
Lesen: 16K RAM oder Erw.- ROM, normalerweise ein FutureOS ROM
Schreiben: 16K Video-RAM bzw. ein ROM der ROM-RAM-Box oder Ähnliches

In diesem Block wird das FutureOS API angesprungen. Einzelne System-Funktionen können hier direkt aufgerufen werden. Schreibt man Daten in diesen Block, so wird das Video-RAM beschrieben.

Das System RAM

Der Bereich der Systemvariablen hat fogenden Aufbau. Es wird der Bereich zwischen &A000 und &BFFF beschrieben. Von oben nach unten:

&BE02 bis &BFFF: 510 Bytes Stack also 255 Elemente (16 Bit)

&BD00 bis &BE00: (&BE00 inclusiv) 257 Bytes INTERUPT MODE 2 Vektoren

&BC80 bis &BCFF: 128 Byte Datei-Header (Hintergrund-Programm)

&BC00 bis &BC7F: 128 Byte Datei-Header (Vordergrund-Programm)

&B800 bis &BBFF: 1024 Bytes als 1 KB Speicher für System-Variablen
(Genauere Aufschlüsselung: Siehe unten)

&B000 bis &B7FF: 2048 Bytes Textbildschirm Puffer. Dieser 2 KB Bereich steht dem Anwender bzw. Programmierer zur freien Verfügung.

&A000 bis &AFFF: Dieser Bereich ist mit Datei - Markierungs – Bytes (File-Tagging-Bytes = FTB) belegt. Diesen bestimmen, ob eine Datei markiert, nur-lesbar, Sys od. Arc. ist.

Wenn das DIR eines Laufwerks gerade nicht eingelesen ist, dann steht der entsprechende Bereich dem Anwender bzw. Programm zur freien Verfügung. Die RAM-Variablen TURBO_A bis TURBO_M zeigen an, ob der RAM Bereich eines Speicher-Mediums genutzt werde kann (DIR gelesen?).

A000-A0FF: 256 tagging Bytes für FDC 0 (intern) und LW A
A100-A1FF: 256 tagging Bytes für FDC 0 (intern) und LW B
A200-A2FF: 256 tagging Bytes für FDC 0 (intern) und LW C
A300-A3FF: 256 tagging Bytes für FDC 0 (intern) und LW D

A400-A4FF: 256 tagging Bytes für FDC 1 (extern) und LW E
A500-A5FF: 256 tagging Bytes für FDC 1 (extern) und LW F
A600-A6FF: 256 tagging Bytes für FDC 1 (extern) und LW G
A700-A7FF: 256 tagging Bytes für FDC 1 (extern) und LW H

A800-A9FF: 512 tagging Bytes für Hard-Disk Partition I
AA00-ABFF: 512 tagging Bytes für Hard-Disk Partition J
AC00-ADFF: 512 tagging Bytes für Hard-Disk Partition K
AE00-AFFF: 512 tagging Bytes für Hard-Disk Partition L

Laufwerk M wird – falls vorhanden – dynamisch gesetzt (LW C, D, G oder H)

Jedes Datei-Markierungs-Byte hat folgenden Aufbau:

Bit 0 - Null => nicht markiert /// Eins => markiert
Bit 1 - Null => nicht zuvor markiert /// Eins => war zuvor markiert
Bit 2 - reserviert
Bit 3 - reserviert
Bit 4 - reserviert
Bit 5 - RW / RO ;===> Aufbau wie im DIR Eintrag (32er Orginal)
Bit 6 - DIR / SYS ;===> Aufbau wie im DIR Eintrag (32er Orginal)
Bit 7 - NoARC / ARC ;===> Aufbau wie im DIR Eintrag (32er Orginal)

Im Turbo-Desktop wird eine markierte Datei unterstrichen gezeigt. Eine zuvor markierte, aber bereits bearbeitete Datei wird durchgestrichen dargestellt.

Die Attribute RO, SYS, ARC werden durch Invertierung des zugehörigen Buchstabens der Datei-Erweiterung (letzte drei Zeichen der 11 Zeichen des Dateinamens) dargestellt.

Erklärung der RAM-Variablen

Im folgenden wird zu jeder RAM-Variable eine kurze Information gegeben. Von manchen Speicherstellen sollte man Anfangs die Finger lassen, da ein falsches Byte am falschen Ort die System-Stabilität beeinflussen kann.

ALLE diese RAM Variablen sind in der Datei **#EQU-API.DEU** zusammengefasst und stehen dem Programmierer zur Verfügung.

Dabei bedeutet der Ausdruck EQU soviel wie "=", also der Ausdruck zur Linken entspricht dem Ausdruck zur Rechten.

DS steht für "define space". Der Wert nach DS gibt an, wieviele Bytes die entsprechende Variable (steht vor DS) belegt.

Abkürzungen: ***LW = Laufwerk***
DIR = DIRectory = Inhaltsverzeichniss (Disk oder HD).

Zwei Floppy Disk Kontroller FDC 765 können an den CPC angeschlossen werden. Der FDC von Amstrad wird als interner FDC bezeichnet, während der FDC von Vortex der externe FDC ist. Die FDCs unterscheiden sich lediglich durch ihre I/O Adressen. Die beiden folgenden Variablen enthalten die Port-Adresse der Haupt-Status-Register beider FDC765:

```
FDC0_ST EQU &FB7E ;Haupt-Status-Register(HSR), interner FDC
FDC1_ST EQU &FBF6 ;HSR, externer FDC(Vortex)
```

Die Variablen im einzelnen

Die folgenden Variablen geben an, ab welcher Adresse die Datei-Markierungs-Bytes eines jeden Speichermediums beginnen. Siehe zuvor.

TMS_A EQU &A000 ==> Laufwerk A -----/<T>agging <M>assen <S>peicher_<A>
TMS_B EQU &A100 ==> Laufwerk B /
TMS_C EQU &A200 ==> Laufwerk C / Für die Laufwerke des internen FDC
TMS_D EQU &A300 ==> Laufwerk D / (der FDC auf der Platine des 6128)

TMS_E EQU &A400 ==> Laufwerk E / Für die Laufwerke des externen FDC
TMS_F EQU &A500 ==> Laufwerk F / (FDC der Vortex F1-D oder M1-D)
TMS_G EQU &A600 ==> Laufwerk G /
TMS_H EQU &A700 ==> Laufwerk H /

TMS_I EQU &A800 ==> Partition I der Dobbertin HD20 Festplatte
TMS_J EQU &AA00 ==> Partition J ""
TMS_K EQU &AC00 ==> Partition K ""
TMS_L EQU &AE00 ==> Partition L ""

TAR16 EQU &1800 ==> Dieser 8 KB Puffer (von &1800 bis &37FF) dient der Umwandlung eines 32er DIR (32 Byte pro Eintrag) auf 16er DIR (16 Byte) durch die Routine DIRWA. Das 32er DIR entspricht dem DIR auf (Hard)Disc. Das 16er DIR wird auf dem Bildschirm angezeigt.

TXT_SCR EQU &B000 ==> Anfang des 2 KB Textbildschirms (z.B. für DUMP)

Nun werden die Variablen von B800 bis BBFF erklärt. Dabei sind die Variablen nach Verwendungszweck geordnet (Adressen siehe Datei **#EQU-API.DEU**).

TAS_S1 DS 64 ==> 64 (32*2) Bytes für Mode 1 Terminal JP Tabelle.
TAS_S2 DS 64 ==> 64 (32*2) Bytes für Mode 2 Terminal JP Tabelle.
Weiteres im Kapittel über die Zeichenausgabe.

RAMCHAR DS 1 ==> Bestimmt den Bildschirm-Modus 0...3 (Bits 1, 0) und selektiert, ob der Zeichensatz des ROMs (Bit 2 = 0) oder des RAMs (Bit 2 = 1) verwendet werden soll. Der Zeichensatz liegt zwischen immer &3800 und &3FFF.

C_POS DS 2 ==> Diese zwei Bytes geben die aktuelle Cursor POSition an. Dabei muß in Mode 1 der Wert 2 und in Mode 2 der Wert 1 addiert werden um die akt. V-RAM Adresse zu ermitteln.

Mode 1: &BFFE bis &C7FE.
Mode 2: &BFFF bis &C7FF.

FDC_RES DS 7 ==> Die sieben Bytes der letzten Result-Phase des FDC 0 (intern, Schneider/Amstrad) und 1 (extern, Vortex) werden hier abgelegt, sie geben Aufschluß über Erfolg oder Mißerfolg der letzten Diskettenoperation.

MO_ST DS 1 ==> Dieses Byte erlaubt die Manipulation des MOtor STatus aller Floppy Disk Laufwerke.

MO_ST ist normalerweise auf &00 gesetzt. Somit sind die Laufwerks-Motoren permanent aus, es sei denn sie werden extra eingeschalten.

Enthält dieses Byte allerdings den Wert &FF, dann laufen alle LW Motoren permanent. Dies spart quasi die Hochlauf-Zeit bzw. Ready-Zeit.

Wie kann man die Motoren EIN oder AUS schalten?

```
Motoren ein:  LD BC,&FA7E : LD A,&FF      : OUT (C),A          ;MSB = 1
Motoren aus:  LD BC,&FA7E                 : OUT (C),C          ;Bit 0=0 => MSB = 0
```

Diese Variablen bestimmen die Anzahl der Lese-/Schreib-Versuche:
FDCLSA DS 1 ==> Aktueller Versuch beim lesen/schreiben FDC 0/1 (variabel)
FDCLSV DS 1 ==> Zahl der Versuche beim lesen/schreiben FDC 0/1 (konstant)

Jedes Laufwerk hat seine spezifische Spurwechselzeit. Die Daten der Spur-Wechsel-Zeit aller acht Laufwerke A...H stehen im RAM ab DSWZ.

Jedes Laufwerk hat eine eigene Ein-Byte Variable, der Inhalt des Low-Nibbles muss immer Null sein.

Laufwerk	A	B	C	D	E	F	G	H
Bytes ab DSWZ	&A0	&A0	&E0	&E0	&E0	&E0	&E0	&E0

```
Laufwerk: A ! B ! C ! D ! E ! F ! G ! H ! Die Spur-Wechsel-Zeit
-----------+---+---+---+---+---+---+---+ Variablen für alle acht
DSWZ DB  &A0,&A0,&E0,&E0,&E0,&E0,&E0,&E0! Floppy-Disk Laufwerke
```

AKT_ROM DS 2 ==> Im Low-Byte steht die Nummer des gerade aktiven ROMs (&00-&FF). Ihm folgt das High-Byte &DF. Dies ermöglicht die folgende Befehlsfolge:

```
LD  BC,(AKT_ROM) ;B = &DF, C = aktuelle ROM Nr.
OUT (C),C        ;selektiertes ROM einblenden
```

AKT_RAM DS 2 ==> Das Low-Byte bestimmt das gerade aktive RAM. Enthält es &C0 so sind die Standart 64 KB eingeblendet. Die Werte &C4 - &FF blenden hingegen ein Erw.-RAM zwischen &4000 und &7FFF ein. Das High-Byte ist &7F. Dies erlaubt folgende Programmierung:

```
LD  BC,(AKT_RAM) ;RAM Konfig. in BC laden
OUT (C),C        ;und RAM einblenden
```

Sind an den CPC mehr als 512 KB E-RAM angeschlossen, so kann das High-Byte &78-&7E anstatt &7F enthalten.

TAST_R0 DS 10 ==> Diese 10 Bytes stellen die PSG Tastaturreihe 0 bis 9 dar. Das sind 10 * 8 = 80 Bit. Jedes gelöschte Bit signalisiert eine gedrückte Taste. Das Bit einer nicht gedrückten Taste ist auf 1 gesetzt. Die Bytes dieser Matrix müssen durch HOLETST gefüllt werden, da die Tastatur nicht durch Interrupts abgefragt wird (um CPU Zeit zu spaaren).

Die folgenden 13 mal acht Bytes sind für die Verwaltung diverser Speichermedien nötig. Jedes Medium (LW, HD, MM) nutzt acht Bytes:

TURBO_A DS 8 ==> Turbo Desk Laufwerk A (Amstrad Kontroller, int. FDC)
TURBO_B DS 8 ==> Turbo Desk Laufwerk B
TURBO_C DS 8 ==> Turbo Desk Laufwerk C
TURBO_D DS 8 ==> Turbo Desk Laufwerk D

TURBO_E DS 8 ==> Turbo Desk Laufwerk E (Vortex Kontroller, ext. FDC)
TURBO_F DS 8 ==> Turbo Desk Laufwerk F
TURBO_G DS 8 ==> Turbo Desk Laufwerk G
TURBO_H DS 8 ==> Turbo Desk Laufwerk H

TURBO_I DS 8 ==> Turbo Desk HD Partition I (Dobbertin HD)
TURBO_J DS 8 ==> Turbo Desk HD Partition J
TURBO_K DS 8 ==> Turbo Desk HD Partition K
TURBO_L DS 8 ==> Turbo Desk HD Partition L

TURBO_M DS 8 ==> Turbo Desk virtuelles Laufwerk M (MM)

Die Bytes jeder 8 Byte-Gruppe enthalten folgende Informationen. Dabei ist ihre Bedeutung bei jedem Speichermedium identisch.

Byte 0 - LW Tagging Byte. Dieses Byte zeigt u.A. ob LW aktiv ist.

Die Bits 7, 6, 5 und 4 enthalten die Format-Nummer des LWs.

&0X ==> HD Partition // noch kein Floppyzugriff // LW hat Fremdformat
&1X ==> die eingelegte Disk hat VORTEX Format.
&2X ==> IBM ---------- Format
&4X ==> SYSTEM ----- Format
&8X ==> FutureOS --- Format
&CX ==> DATA -------- Format

Das Bit 3 zeigt an, ob das Inhaltsverzeichniss bereits manipuliert wurde, dies geschieht z.B. beim Umbenennen einer Datei. Dieses Bit ist nur dann relevant wenn Bit 0 = 1 ist, also LW aktiv.

Bit 3 = 0 ==> DIR wurde noch nicht manipuliert.
Bit 3 = 1 ==> DIR wurde durch Datei-Operation manipuliert.

Das Bit 2 gibt Aufschluß über die gerade aktive Kopf-Nummer. Dieses Bit ist nur bei einseitigen Formaten relevant, die auf doppelseitigen LWs betrieben werden. Bei Vortex-Format sollte es immer = 0 sein.

Bit 2 = 0 ==> Kopf-Nummer 0 wird benutzt (Standart).
Bit 2 = 1 ==> Kopf-Nummer 1 wird benutzt (Format-abhängig!)

Das Bit 1 gibt an ab das LW überhaupt vorhanden und aktiv ist. Ein LW, daß nicht vorhanden oder aktiv ist, wird im Desktop schraffiert dargestellt.

Bit 1 = 0 ==> Das LW ist vorhanden und betriebsbereit.
Bit 1 = 1 ==> Das LW ist nicht vorhanden bzw. inaktiv.

Das Bit 0 zeigt ob das LW markiert ist, oder nicht. Man kann nur mit markierten Laufwerken arbeiten, die anderen werden nicht beachtet.

Bit 0 = 0 ==> Das LW ist nicht markiert, es wird ignoriert. Die Bits von 1 bis 7 sind deshalb bedeutungslos.
Bit 0 = 1 ==> Das LW ist markiert, man kann damit arbeiten.

Nun folgen die restlichen sieben Bytes, sie sind jedoch lediglich dann relevant, wenn das entsprechende Laufwerk auch als aktiv markiert ist.

Byte 1 - RAM Block &C4, ..., &FF oder &C0. (32er DIR)
Byte 2 - StartAdresse (in Pages) &40 - &7F. (32er DIR)
Byte 3 - Länge (in Pages) &00 - &40. (32er DIR)

Byte 4 - RAM Block &C4, ..., &FF oder &C0. (16er DIR)
Byte 5 - StartAdr (in Pages) &40 - &7F. (16er DIR)
Byte 6 - Länge (in Pages) &00 - &20. (16er DIR)

Byte 7 - Anzahl der 64 Einträge Pages (z.B. = 1 bei 64 Einträgen)

Dabei entspricht das 32er DIR dem Inhaltsverzeichniss auf Diskette, es wird im RAM gepuffert.

Das 16er DIR ist eine Textversion des 32er DIRs. Es dient dazu, um es auf dem Bildschirm anzuzeigen, so wie es im Desktop eben geschieht.

TURBO_X DS 2 ==> Das Low-Byte enthält die Nummer des obersten freien Erweiterungs-RAMs &C4,..,&FF, alle E-RAMs darüber sind durch RAM gepufferte DIRs belegt. Enthält es den Wert &C0, dann ist das E-RAM verbraucht.

Das High-Byte enthält die höchste freie Adresse dieses E-RAMs (in Pages) &41 bis &80.

Beispiel: Byte &78 steht für Adresse &7800, das RAM unterhalb ist also noch frei.

TURBO_X enthält z.B.: &FF, &80 = &80FF wenn 512 KB E-RAM and den CPC angeschlossen sind und noch kein Inhaltsverzeichnis eingelesen ist.

Im Maschinen-Monitor kann man alle Z80-Register einstellen, um danach eine Routiene aufzurufen. Diese Bytes sind an sonsten frei verwendbar.

```
REG_AF1 DS 2 ==> Register AF ==> Erster Register Satz
REG_BC1 DS 2 ==> Register BC
REG_DE1 DS 2 ==> Register DE
REG_HL1 DS 2 ==> Register HL

REG_AF2 DS 2 ==> Register AF' ==> Zweiter Register Satz
REG_BC2 DS 2 ==> Register BC'
REG_DE2 DS 2 ==> Register DE'
REG_HL2 DS 2 ==> Register HL'

REG_IX DS 2    ==> Register IX ==> Index Register
REG_IY DS 2    ==> Register IY

REG_SP DS 2    ==> Register SP ==> Spezial Register Stackpointer
REG_R DS 1     ==> Register R
REG_I DS 1     ==> Register I ==> Achtung: Den Wert &BD nie verändern!
REG_PC DS 2    ==> Register PC
```

Die folgenden Variablen stehen dem Anwender ebenfalls zur freien Verfügung, sie werden als User-Register bezeichnet. Sie dienen größtenteils dazu, Parameter an bestimmte Routinen zu übergeben. Die REG??_? Variablen sollte man allerdings nur benutzen um kurzzeitig(!) Werte zu speichern. Denn viele Routinen verändern diese Variablen.

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

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

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

Das OS muss wissen, wieviele Dateien sich auf einer Diskette befinden. Folgende Variablen geben die Dateien-Zahl eines jeden Laufwerks an. Je nach Format sind das 0 bis 64 bis 128 bis 512 Einträge im DIR.

TMD_A DS 2 ==> <T>agging <M>assenpeicher <D>ateienanzahl <A>. (intern)
TMD_B DS 2
TMD_C DS 2
TMD_D DS 2

TMD_E DS 2 ==> FDC extern, also Vortex F1-S, F1-D od. Dobbertin D-DOS.
TMD_F DS 2
TMD_G DS 2
TMD_H DS 2

TMD_I DS 2 ==> HD 20MB z.B. die 20 MB Dobbertin Platte
TMD_J DS 2
TMD_K DS 2
TMD_L DS 2

TMD_M DS 2 ==> Memory Floppy

Diese folgenden acht Bytes stellen die Daten der Echtzeit-Uhr dar. Die ROM Version des SETUP enthält hier Brenn-Datum und -Zeit des Eproms.

UHR_00	DS 1 ==> Byte 0: Hundertstel und Zehntel Sekunden
UHR_SEK	DS 1 ==> Byte 1: Sekunde
UHR_MIN	DS 1 ==> Byte 2: Minute
UHR_STU	DS 1 ==> Byte 3: Stunde
UHR_WOT	DS 1 ==> Byte 4: Wochentag. Bit 5 löschen um RTC zu starten!
UHR_TAG	DS 1 ==> Byte 5: Tag
UHR_MON	DS 1 ==> Byte 6: Monat
UHR_JAR	DS 1 ==> Byte 7: Jahr (Zehner und Einer, z.B.: 9 = 2009)

UHR_ROM DS 2 ==> Das Low-Byte gibt die ROM Nummer der Dobbertin/dxs RTC bzw. Echtzeituhr der M4 Karte an. Der ROM-Nummer der Echtzeituhr folgt das High-Byte &DF um folgende Befehle zu ermöglichen:

```
LD  BC,(UHR_ROM) ;B = &DF, C = ROM Nr. der Dobb. RTC
OUT (C),C        ;Uhr (als ROM) einblenden
```

Die beiden folgenden Variablen geben die aktuelle Ausdehnung des Video-RAMs an. Unabhängig vom Bildschirmmodus werden immer die Mode 2 Werte verwendet.

Diese Werte dürfen auf keinen Fall überschritten werden, wenn der Cursor positioniert wird. Sonnst könnten Bytes irgendwo ins RAM geschrieben werden. In den Parametern von Control-Code muss darauf Rücksicht genommen werden.

MAX_CRX DS 1 ==> Horizontale(X) Spaltenanzahl, maximal 104
MAX_CRY DS 1 ==> Vertikale(Y) Zeilenanzahl, maximal 40

MAX_RAM DS 1 ==> Enthält die Anzahl aller 16K Erweiterungsblöcke, die frei und lückenlos ab &7FC4 genutzt werden können. Aber bitte die XRAM Variablen überprüfen.

ADROM_X DS 1 ==> 8 Bit Nummber (ROM - Auswahl) eines Erweiterungs-ROMs für FutureOS. E-ROM welches zuletzt das OS benutzte.

ADROM_A DS 2 ==> Nummer des FutureOS ROM A, gefolgt vom Byte &DF. Dies erlaubt: **`LD BC,(ADROM_A):OUT (C),C`** um FutureOS ROM A einzublenden.

FREEZE DS 1 ==> Das oberste Bit (Bit 7) wird für das USB Token der Albireo USB Maus benutzt. Die Bits 6 bis 0 sind bisher ungenutzt.

DIRIN DS 1 ==> zeigt an, ab ein DIRectory eINgelesen wurde. Die Variable enthält entweder die Kennung des Speichermediums A...M = &00 bis &0C, oder den Wert &FF. Bei &FF wurde kein DIR gelesen, die LW-Tag-Bytes TURBO_A - M haben also keine Bedeutung.

Die 12 Konfig / Setup Bytes enthalten Informationen über diverse Eigenschaften des OS. Genaueres ist in Datei **#OS-VAR.DEU** zu finden.

KF_MED DS 1 ==> gibt Auskunft darüber, ober der SPARtan Modus aktiv ist, ober Zahlen im hexadezimal oder dezimal System eingegeben werden und ob PlayCity, MultiPlay, CPC Digiblaster, Albireo, M4 board oder ein VN96 WizCat Netzwerkadapter angeschlossen ist.

KF_FDHD DS 1 ==> zeigt ob FDC intern oder extern vorhanden, ob eine Dobbertin HD20 oder Vortex Winchester angeschlossen ist. Ob eine Dobbertin/dxs / Happy Computer Echtzeituhr oder ein Kassettenlaufwerk vorhanden ist. Auch die Existenz des CPC Boosters ist vermerkt.

KF_AB DS 1 ==> informiert über Spur- & Seiten- Zahl der Laufwerke A und B. Info über Fabrikat und Verfügbarkeit.

KF_CD DS 1 ==> "" ... der Laufwerke C und D.

KF_EF DS 1 ==> informiert über Spur- & Seiten- Zahl der Laufwerke E und F. Info über Fabrikat und Verfügbarkeit.

KF_GH DS 1 ==> "" ... der Laufwerke G und H.

KF_MEM DS 1 ==> Zeigt welche 512 KB Blöck innerhalb den maximal möglichen 4 MB E-RAM angeschlossen sind.

KF_SIO DS 1 ==> informiert darüber welche seriellen Schnittstellen angeschlossen sind, und über 7 oder 8 - Bit Druckerport, sowie über Farb- bzw. Monochrom- Monitor.

KF_CPC DS 1 ==> Hierin ist der CPC Typ codiert, auf dem das FutureOS läuft. Die Sprache des Users ist definiert. Sind X-MASS/SF2 vorhanden?

KF_VERS DS 1 ==> Definiert FutureOS Version (PD oder personalisiert), Darstellung von Zeit und Datum (Icons bzw. Ziffern), Auto-DIR und denSprachsynthesizer (LambdaSpeak, SSA-1 oder dk'tronics).

KF_OSUN DS 2 ==> Persönliche Serien- / Benutzer-Nummer des Anwenders.

Folgende fünf Variablen/Bytes werden von den Routinen SC_AB, SC_AU und SCRI benutzt. Sie dienen dem Desktop DIRectory-Seiten anzuzeigen:

TDRAM	DS 1	==> Der RAM Block des 16er DIRs des aktuellen LWs
TDHST	DS 1	==> High-Byte der Startadresse des 16er DIRs
TDANZ	DS 1	==> Anzahl 1K Seiten innsgesammt
TDAKT	DS 1	==> Aktuelle 1K Seite
TDLWK	DS 1	==> Aktuelles Laufwerk

In FutureOS existieren vier Tastatur-Ebenen. Diese vier mal 80 Bytes bestimmen den Wert (&00-&FE) einer gedrückten Taste:

TAST_N DS 80 ==> 80 Bytes für Tastaturmatrix 10 * 8 ==> NORMAL
TAST_S DS 80 ==> 80 Bytes für Tastaturmatrix 10 * 8 ==> SHIFT
TAST_C DS 80 ==> 80 Bytes für Tastaturmatrix 10 * 8 ==> CONTROL
TAST_SC DS 80 ==> 80 Bytes für Tastaturmatrix 10 * 8 ==> SHIFT/CONTROL

Die ersten 512 KB Erweiterungs RAM

An den CPC kann man bis zu 4 MB Erweiterungs-RAM anschließen. Den ersten 512 KB kommt unter FutureOS besondere Bedeutung zu: Dieses RAM wird in Form von 16 KB Blöcken verwaltet, einzelne Blöcke werden zwischen &4000 und &7FFF eingeblendt.

Für jeden diese 32 Blöcke a 16 KB ist eine 1 Byte Variable zuständig, die darüber Auskunft gibt, ob der Block vorhanden ist und wie er verwendet wird. Diese Variablen sind folgendermaßen konstruiert:

Bit 7 ==> = 1 ==> RAM ist durch geladene Datei belegt.
Bit 6 ==> = 1 ==> RAM wird durch Multitasking Routine belegt.
Bit 5 ==> = 1 ==> ??? reserviert!!!
Bit 4 ==> = 1 ==> RAM wird vom User belegt. OS ignoriert Block.
Bit 3 ==> = 1 ==> RAM wird als KURZZEIT Puffer verwendet.
Bit 2 ==> = 1 ==> RAM wird als LANGZEIT Puffer verwendet.
Bit 1 ==> = 1 ==> RAM wird als DIR Puffer verwendet.
Bit 0 ==> = 0 ==> Ram nicht vorhanden. /// = 1 ==> RAM existiert.

Dabei darf von den Bits 1..7 nur jehweils EINES gesetzt sein, das über den Verwendungszweck informiert.

Dies sind die Namen der 32 E-RAM Variablen:

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

Die Endungen der Variablen _C4 ... _FF entsprechen der physikalischen E-RAM Auswahl (low byte xx) über Port &7Fxx.

Mehr als 512 KB E-RAM

Das Erweiterungs-RAM des CPC lässt sich in 512 KB große Blöcke einteilen. Von diesen Blöcken wird je einer über die Port-Adresse &7Fxx (erste 512 KB E-RAM), &7Exx, &7Dxx, &7Cxx, &7Bxx, &7Axx, &79xx oder &78xx angesprochen.

Das Low-Byte wird dabei für jeden 512 KB Block so verwendet, wie es bei den ersten 512 KB E-RAM (&7Fxx) üblich ist. Auf diese Weise lassen sich bis zu 4 MB E-RAM verwalten.

Sind an den CPC mehr als 512 KB Erweiterungs-RAM angeschlossen, so wird dieser zusätzliche Erweiterungs-Speicher durch die folgenden Variablen verwaltet.

FutureOS initialisiert diese Variablen beim System-Start.

X4RXE DS 1 ==> Diese Variable zeigt, ob überhaupt mehr als 512 KB Erweiterungs-RAM vorhanden sind oder nicht:
Ist Bit 7 = 0 dann sind maximal 512 KB E-RAM vorhanden.
Ist Bit 7 = 1 dann sind mehr als 512 KB E-RAM am CPC angeschlossen.

Das Bit 4 hat eine Spezialfunktion: Ist es gesetzt dann können über Port &7ECx genau 64 KB angesprochen werden (&C4-&C7). Dies ist beim X-MEM Fall, beim CPC6128 bzw. 6128 Plus kann das interne E-RAM auf diese Weise angesprochen werden, so daß insgesamt 640 KB verfügbar sind.

Die Bits 3, 2, 1 und 0 der Variable X4RXE zeigen an, ob durch die Ports &7EFx, &7EEx, &7EDx und &7ECx E-RAM adressierbar ist.

Dabei bedeutet ein auf 0 gelöschtes Bit, dass kein E-RAM vorhanden ist. Ist das Bit jedoch auf 1 gesetzt, so ist der entsprechende 128 KB Block vorhanden. Enthält Variable X4RXE den Wert Null, so sind maximal 512 KB E-RAM angeschlossen.

Die folgende Tabelle gibt Aufschluss, welche E-RAM Blöcke oberhalb der ersten 512 KB E-RAM vorhanden sind. Je ein Bit einer der vier Variablen symbolisiert 128 KB E-RAM, das über den Port &7NNx angesprochen wird. X4RXE Bit 4 symbolisiert 64 KB (&7Exx: &C4-&C7).

Bit \ Variable	7	6	5	4	3	2	1	0
X4RXE	512KB+?	-	-	&7EC4-7	&7EFx	&7EEx	&7EDx	&7ECx
X4RDC	&7CFx	&7CEx	&7CDx	&7CCx	&7DFx	&7DEx	&7DDx	&7DCx
X4RBA	&7AFx	&7AEx	&7ADx	&7ACx	&7BFx	&7BEx	&7BDx	&7BCx
X4R98	&78Fx	&78Ex	678Dx	&78Cx	&79Fx	&79Ex	&79Dx	&79Cx

Ein gelöschtes Bit bedeutet, daß die betreffenden 128 KB E-RAM nicht vorhanden sind.
Ist das Bit jedoch gesetzt, so sind auch die entsprechenden 128 KB E-RAM vorhanden.
Weitere Informationen sind in Datei **#OS-VAR.DEU** zu finden.

Diese vier Variablen unterteilen die max. 4 MB E-RAM logisch in 128 KB Blöcke, unter FutureOS wird das gesamte E-RAM allerdings in Blöcken von 16 KB verwaltet, von denen jeweils einer zwischen &4000 und &7FFF eingeblendet wird.

Ab X4RAM liegen 28 Bytes im System-RAM des CPC. Die 224 Bits dieser 28 Bytes verwalten E-RAM das über die ersten 512 KB E-RAM hinausgeht. Die ersten 512 KB werden hierbei nicht beachtet, da sie ja in den XRAM Variablen verwaltet werden.

Jedes der 224 Bits entspricht einem 16 KB E-RAM Block. Ist das Bit gelöscht, dann wird das RAM gerade nicht benutzt. Ist das Bit hingegen auf 1 gesetzt, dann wird das zugehörige E-RAM momentan benutzt.

Folgende Tabelle zeigt welches Bit zu welchem 16 KB E-RAM Block zugehörig ist:

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

Im FutureOS Monitor kann man die RAM- und ROM- Konfiguration einstellen, folgende Variablen enthalten den aktuellen Status.

MON_ROM DS 1 ==> &82 ==> Unteres ROM eingeblendet // &86 kein U.ROM
MON_RAM DS 1 ==> Nummer des 16KB E-RAM Blocks &C0,&C4,..,&FF
MON_MMB DS 1 ==> &A0 ==> Memory mapped I/O inaktiv // &B8 ==> MM aktiv

Achtung: Nur der CPC Plus hat Memory Mapped I/O.

Die OS Funktion XWART benutzt Variablen um die die Wartezeiten zu definieren. Genauere Dokumentation findet sich in Datei API-A-DE.DOK.

VZ_MOD DS 1 ==> VerZögerung MODus = 0 dann Anfangs-VZ, sonst Folge-VZ.
VZ_AG DS 2 ==> VerZögerung Anfang Generell (Wiederladewert)
VZ_AA DS 2 ==> VerZögerung Anfang Aktuelle Restzeit
VZ_FG DS 2 ==> VerZögerung Folge Generell (Wiederladewert)
VZ_FA DS 2 ==> VerZögerung Folge Aktuelle Restzeit

DEFINT DS 1 ==> Variable definiert die Interrupts ein- (EI) oder ausgeschalten (DI) sind. Achtung: FutureOS arbeitet im Interrupt-Modus 2! Hat DEFINT den Wert &00, dann sind die Interrupts aus (DI). Hat sie dagegen den Wert &FF, dann sind die Ints. aktiv (EI).

Die Interrupts sind üblicherweise ausgeschalten, dadurch stehen sowohl der IM 1 als auch der IM 2 Einsprung den Applikationen zur Verfügung.

M4_ERAM DS 1 ==> Definiert einen 16 KB E-RAM Block innerhalb der ersten 512 KB des Erweiterungsspeichers (&C4 - &FF). Dieser Block wird zur Verwaltung der SD-Karte der M4 Erweiterung verwendet.

IDE_XR DS 1 ==> Definiert die ROM Nummer des XROMs welches beim Klick auf das IDE- / Massenspeicher-Icon aufgerufen wird (Adresse &C009). Bei &00 ist kein solches XROM vorhanden. Das XROM selbst setzt diese Variable.

Durch das OK Icon ist es möglich vom Desktop zu einem Programm zurückzukehren. Das OK Icon hat einige eigene 16 Bit Variablen:

OK_ADR DS 2 ==> ADResse die durch das OK Icon aufgerufen wird &????
OK_ATE DS 2 ==> Inhalt muss mit (OK_ADR) ADDiert &FFFF ergeben
OK_BLK DS 2 ==> 16K RAM bei Aufruf von OK &C0,&C4,..,&FF einblenden.
OK_BTE DS 2 ==> Inhalt muss mit (OK_BLK) ADDiert &FFFF ergeben

CAPS DS 1 ==> Definiert den Caps-Lock Modus. Bei CAPS = &00 wird jeder Tastendruck normal dargestellt. Enthält CAPS aber &FF, dann werden alle kleinen Buchstaben GROSS gezeigt.

Die folgenden 16 Bit Variablen werden vom I Icon genutzt:
I_ADR DS 2 ==> ADResse die bei I Icon aufgerufen wird &????
I_ATE DS 2 ==> Inhalt muss mit (I_ADR) ADDiert &FFFF ergeben
I_BLK DS 2 ==> BLocK bei Aufruf von I an &7F00 + &C0,&C4,&C5,..,&FF
I_BTE DS 2 ==> Inhalt muss mit (I_BLK) ADDiert &FFFF ergeben

Diese beiden Variablen dienen der Verwaltung des Drucker-Puffer:
DRUMEM DS 1 ==> phys.Block-Nr. erster E-RAM Block Spooler &C4,..,&FF
DRUBAZ DS 1 ==> Anzahl direkt folgende E-RAM Blocks 0..31 f. Spooler

Eine mögliche RAM-Floppy wird von diesen Variablen verwaltet:
MMFMEM DS 1 ==> phys.Block-Nr. erster E-RAM Block RAM-Floppy &C4,..,&FF
MMFBAZ DS 1 ==> Anzahl direkt folgende E-RAM Blocks 0..31 f. RAM-Floppy

HI_MEM DS 2 ==> Zeiger, oberstes freies Byte, OS-Start, normal &9FFF.
Diese Variable entspricht dem HIMEM des Basic. (Vom OS nicht benutzt).
Falls eine Applikation RAM belegt, sollte HI_MEM angepasst werden.

AUH_00 DS 1 ==> 8 Bytes, analog UHR_Daten
AUH_SEK DS 1 ==> geben **alte** Zeit an
AUH_MIN DS 1 ==> vor letztem Lesen
AUH_STU DS 1
AUH_WOT DS 1
AUH_TAG DS 1
AUH_MON DS 1
AUH_JAR DS 1

WECK_ST DS 1 ==> = &00 ==> keine Weckzeit // = &FF ==> Alaram aktiv
WECK_ZS DS 1 ==> Sekunde Weckzeit
WECK_ZM DS 1 ==> Minute
WECK_ZH DS 1 ==> Stunde

BILD_SS DS 2 ==> Start-Adresse des Bildschirm-Schoners
BILD_RB DS 1 ==> RAM-Block des Bildschirm-Schoners
BILD_ST DS 1 ==> = &00 ==> kein Bild-Schoner // = &FF ==> BSS aktiv

BSS_WW DS 2 ==> Bild-Schirm-Schoner Wiederlade-Wert &0000-&FFFF
BSS_WA DS 2 ==> Bild-Schirm-Schoner Wert aktuell, kleiner als BSS_WW

HGB_RB DS 1 ==> RAM &C4-&FF, 16 KB Mode 2 Hintergrundbild f. Desktop!
HGB_ST DS 1 ==> HinterGrundBild Status, &00 => kein Bild, &FF => Bild

L_RAM DS 1 ==> &00 => Unteres RAM nicht gepuffert, oder &C0-&FF => physikalisches RAM indem als L-RAM (&0000-&3FFF) gepuffert ist.

BORDER DS 1 ==> Hardware-Farb-Wert Rand
INK_0 DS 1 ==> Hardware-Farb-Wert INK 0 (Untergrund)
INK_1 DS 1 ==> Hardware-Farb-Wert INK 1 (Stift 1)
INK_2 DS 1 ==> Hardware-Farb-Wert INK 2 (Stift 2)
INK_3 DS 1 ==> Hardware-Farb-Wert INK 3 (Stift 3)

FDC_ERR DS 2 ==> Adresse FDC765-Fehler-Behandler
FDE_RAM DS 2 ==> Fehler-Behandler-RAM &C0-&FF, gefolgt von &7F

JT_JH DS 1 ===> Diese Variable enthält das aktuelle Jahrtausend (oberes Nibble) und das aktuelle Jahrhundert (unteres Nibble). Variable JT_JH ist BCD codiert, so steht z.B. der Wert &20 für eines der Jahre 20??. Nach Ablauf des Jahres 9999 muss ein Update des OS durchgeführt werden.

F2AMS DS 1 ===> Enthält diese Variable den Wert &FF so wurde FutureOS von AmsDOS aus als Unterprogramm aufgefufen. In diesem Fall wurden die ersten 48 KB (ohne Bildschirmspeicher) in das E-RAM gesichert. Enthält diese Variable jedoch den Wert &00, so wurde FutureOS regulär aufgerufen.

HEGP2 DS 1 ===> Das Bit 0 zeigt an, ob das Hegotron Grafpad 2 am CPC angeschlossen ist (Bit 0 = 1) oder nicht (Bit 0 = 0). Bit 1 enthält den Status des Grafpads. Ist es auf 0 gelöscht, so ist das Grafpad inaktiv. Ist es auf 1 gesetzt, so ist das Grafpad AKTIV.

Die folgenden System-Variablen verwalten das CPC-IDE, den IDE-Teil des SYMBiFACE II und das IDE8255. Diese System Variablen werden vom IDE-ROM verwaltet. Es können Hard-Discs bis zu einer maximalen Kapazität von 128 Peta Byte verwaltet werden.

IDE DS 1 ==> Variable IDE zeigt an, ob und welche IDE Geräte an die vorhandenen IDE-Erweiterungskarten angeschlossen sind. Durch diese Variable läßt sich auch selektieren, welchen IDE Adapter man verwenden will (IDE8255 bzw. CPC-IDE / SYMBiFACE II) und ob man die Master bzw. die Slave Device benutzen will. Ausserdem zeigt Variable IDE, ob LBA28 oder LBA48 verwendet wird. Ist Variable IDE = &00, so ist kein IDE Geraet vorhanden. Die einzelnen Bits sind folgendermaßen aufgeschlüsselt:

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

Bit 6 = 0 --> kein IDE8255
      = 1 --> IDE8255 angeschlossen

Bit 5 = 0 --> kein CPC-IDE/SF2
      = 1 --> CPC-IDE/SF2 angeschlossen

Bit 4 = 0 --> IDE Geraet 0 selektiert (siehe Bit 7)
      = 1 --> IDE Geraet 1 selektiert (siehe Bit 7)

Bit 3 = 0 --> CPC-IDE/SF2 Gerät 0 arbeitet mit LBA 28 (Master)
      = 1 --> CPC-IDE/SF2 Gerät 0 arbeitet mit LBA 48 (Master)

Bit 2 = 0 --> CPC-IDE/SF2 Gerät 1 nutzt LBA28 (Slave)
      = 1 --> CPC-IDE/SF2 Gerät 1 nutzt LBA48 (Slave)

Bit 1 = 0 --> IDE8255 Gerät 0 nutzt LBA28 (Master)
      = 1 --> IDE8255 Gerät 0 nutzt LBA48 (Master)

Bit 0 = 0 --> IDE8255 Gerät 1 nutzt LBA28 (Slave)
      = 1 --> IDE8255 Gerät 1 nutzt LBA48 (Slave)
```

IDE_DEV DS 1 ==> Diese Variable zeigt, ob ein bestimmtes IDE Gerät aktuell betriebsbereit ist oder nicht. Ist sie auf &00 gesetzt, so ist kein betriebsbereiters IDE Geraet angeschlossen.
Die einzelnen Bits sind folgendermaßen aufgeschlüsselt:

Die Bits 7, 6, 5 und 4 sind reserviert und auf Null gesetzt.

```
Bit 3 = 0 --> CPC-IDE/SF2 Geraet 0 absent (Master)
      = 1 --> CPC-IDE/SF2 Geraet 0 betriebsbereit

Bit 2 = 0 --> CPC-IDE/SF2 Geraet 1 absent (Slave)
      = 1 --> CPC-IDE/SF2 Geraet 1 betriebsbereit

Bit 1 = 0 --> IDE8255 Geraet 0 absent (Master)
      = 1 --> IDE8255 Geraet 0 betriebsbereit

Bit 0 = 0 --> IDE8255 Geraet 1 absent (Slave)
      = 1 --> IDE8255 Geraet 1 betriebsbereit
```

IDE_RAM DS 1 ==> Enthält den physikalischen RAM - Select (Port &7Fxx) eines 16 KB E-RAMs (&C4 - &FF), das als Hard-Disc Cache verwendet wird. Die Verwendung dieses Cache RAMs beschleunigt den Betrieb der Hard-Disc.

IDE_SECNT DS 1 ==> Diese Variable bestimmt den sogenannten Sector Count (sie wird bisher noch NICHT benutzt)

Die folgenden sechs Variablen adressieren den aktuell zu bearbeitenden 512 Byte langen LBA Sektor einer IDE Hard-Disc. Dabei benutzt LBA28 lediglich die vier Variablen IDE_00_07 bis IDE_24_31, während LBA48 alle sechs Variablen IDE_00_07 bis IDE_40_47 nutzt. Bei der Verwendung von LBA48 kann das FutureOS Hard-Discs mit bis zu 128 PB (Peta Byte) verwalten, das sind 134217728 GB.

IDE_00_07 DS 1 ;LBA28 nutzt IDE_00_07 bis IDE_24_31 (4 Bytes)
IDE_08_15 DS 1
IDE_16_23 DS 1
IDE_24_31 DS 1
IDE_32_39 DS 1 ;LBA48 nutzt IDE_00_07 bis IDE_40_47 (6 Bytes)
IDE_40_47 DS 1

ACHTUNG: Manche Variablen werden vom Desktop nicht benutzt, ein Programm ist aber in jedem Fall daran gebunden!! Nur so ist ein kooperatives Betriebssystem zu relisieren. Da unter FutureOS mehrere Programme zugleich betrieben werden können, ist unbedingt nötig auf die System-Variablen Rücksicht zu nehmen, dies betrifft vorallem die Speicherverwaltung mit den XRAM_?? Variablen!

Variablen in den ROMs des FutureOS

Das ROM A des FutureOS enthält an Adresse &C114 die Variable KOBYA.
Dabei steht KOBYA für 'KOntrollBYte A'
Sie dient der Konfiguratioin des OS. Da KOBYA im ROM liegt muss auch das ROM A verändert werden um KOBYA anzupassen.

Welche Parameter werden nun von KOBYA definiert?

Bit 0 = 0 ==> Joystick 2 wird im Desktop abgefragt um Pfeil zu bewegen
= 1 *=> Keine Joystick 2 Abfrage. Kotkeys sind besser nutzbar

Bit 1 = 0 ==> LambdaSpeak spricht als Mann, männliche Stimme
= 1 *=> LambdaSpeak spricht als Frau, weibliche Stimme

Bit 2 = 0 ==> SF2 RTC falls vorhanden auf BCD umstellen. OS schneller
= 1 *=> SF2 RTC nicht umstellen! OS ist kompatibler zu Anderem!

Bit 3 = 0 ==> LambdaSpeak unter FutureOS stumm, keine Sprachausgabe
= 1 *=> LambdaSpeak spricht OS Meldungen

Bit 4 = 0 ==> Normaler / Kompatibler Modus, WiFi an, Sommerzeit etc.
= 1 *=> Gesundheits-Modus, d.h. WiFi aus, Sommerzeit aus etc.

Die Bits 5 bis 7 werden aktuell nicht benutzt.

Ein '*' markiert die voreingestellte Option.
Im FutureOS enthält die ROM Variable KOBYA normalerweise den Wert &1F = xxx1 1111

An Adresse &C18B befindet sich die Variable KOBYB ('KOntrollBYte B'). Auch sie dient der Konfiguratioin des OS. Welche Parameter werden nun von KOBYB definiert?

Bit 0 = 0 *=> MultiPlay Port 1 nutzt Joystick
= 1 ==> MultiPlay Port 1 nutzt Maus

Bit 1 = 0 *=> MultiPlay Port 2 nutzt Joystick
= 1 ==> MultiPlay Port 2 nutzt Maus

Die Bits 2 bis 7 werden aktuell nicht benutzt.

Ein '*' markiert die voreingestellte Option. KOBYB enthält den Wert &00 = xxxx xx00.

Einblenden der einzelnen FutureOS ROMs A bis D

Die Routinen, die das FutureOS zur Verfügung stellt, sind auf die vier ROMs A, B, C und D verteilt.

Wenn man eine OS-Routine verwenden will, so kann man das API benutzen oder man muss vor Aufruf der Routine das OS-ROM einblenden, in dem sich die entsprechende Routine befindet. Es sei denn, das korrekte ROM ist bereits eingeblendet. Intelligente Programmierung kann hier einige Mikrosekunden spaaren.

Einblenden eines der OS ROMs A, B, C oder D mittels OS Funktionen:

Einblenden von ROM A: `CALL OSRON_A` ;ROM A einblenden

Einblenden von ROM B: `CALL OSRON_B` ;ROM B einblenden

Einblenden von ROM C: `CALL OSRON_C` ;ROM C einblenden

Einblenden von ROM D: `CALL OSRON_D` ;ROM D einblenden

Einblenden der ROMs durch Z80 Befehle:

Einblenden von ROM A:
```
LD  BC,(&FF01) ;physikalische ROM Nummer in C
OUT (C),C ;B=&DF. Nun ROM A einblenden
```

Einblenden von ROM B:
```
LD  BC,(&FF07) ;phys. ROM Nr. von B
OUT (C),C ;ROM einblenden
```

Einblenden von ROM C:
```
LD  BC,(&FF0D) ;phys. ROM Nr. von C
OUT (C),C ;ROM einblenden
```

Einblenden von ROM D:
```
LD  BC,(&FF13) ;phys. ROM Nr. von D
OUT (C),C ;ROM einblenden
```

Das Applikations – Programm – Interface (API)

Das Applikations-Programm-Interface (API) der meisten Betriebssysteme ist relativ langsam. Der schnellste Weg eine OS Funktion zu benutzen ist der direkte Aufruf. Das API des FutureOS unterstützt genau diese Vorgehensweise, es ist einfach und effizient konstruiert, aber vor allem ist es frei von Ballast gehalten.

Um eine OS Funktion in einem der vier OS-ROMs aufzurufen kann man den direkten Weg gehen. Oder man benutzt das API. Das API bietet eine Auswahl an Einsprüngen an. Man kann die folgenden Einsprünge benutzen um das ROM der OS-Funktion vor ihrem Aufruf zu selektieren:

ROM_A, ROM_B, ROM_C bzw. ROM_D.

Dabei wird zuerst das OS ROM einbeblendet in dem sich die Routine befindet, dann wird die Routine aufgerufen (die Zieladresse der Routine ist im Register HL der Z80 enthalten), abschließend erfolgt der Rücksprung. Das neue ROM bleibt eingeblendet.

Die API Einsprünge ROM_Q2Z erlauben es für die Dauer eines Unterprogramms ein bestimmtes OS ROM einzublenden. Q(elle) und Z(iel) stehen dabei für die ROMs A, B, C oder D.

Beispiel: Der API Einsprung ROM_A2C ruft ein Unterprogramm in OS ROM C auf, und kehrt dann mit eingeblendetem OS ROM A zurück. Die Adresse der Ziel-Routine muss dabei in Register IX übergeben werden.

Weitere Informationen über das API können in der Datei API-DEU.DOK nachgelesen werden. Bitte nachschlagen!

Neben den OS ROMs A, B, C und D unterstützt FutureOS auch zusätzliche Erweiterungs-ROMs (E-ROMs). Ein Beispiel ist das IDE-ROM.

FutureOS Erweiterungs ROMs (XROMs)

Diese ROMs sind ähnlich wie Erweiterungs ROMs für das native OS aufgebaut. Von &FF00 bis &FFFF entsprechen sie FutureOS ROM A. Ausserdem enthalten sie von &FD80 bis &FDE3 Funktionen die den XROMs Zugriff auf OS Funktionen gestatten.

XROMs benötigen normalerweise keine Initialisierunsfunktion, können jedoch eine enthalten. Siehe "API-X-DE.DOK" für weitere Informationen. FutureOS XROMs beginnen immer(!) mit den Bytes &02, &09 (ab &C000). Zur Installation von XROMs verwenden Sie bitte nur ROManager 2.16 oder höher.

Programm-Architektur

Ein Programm unter BASIC/AMSDOS hat eine maximale Länge von ca. 42 KB. Unter dem 63 K CP/M oder CP/M Plus kann ein Programm sogar bis zu ca. 62 KB groß werden. Größere Programme müssen Teile nachladen.

Unter FutureOS kann ein Programm am Stück bis 4 MB lang sein. Seine Länge hängt nur vom angeschlossenen Erweiterungs RAM ab. Man sollte sich aber mit etwas weniger als 4 MB begnügen, um die Puffer des OS nicht zu überschreiben (die Puffer benötigen ca. 32 KB).

* Unter FutureOS existieren zwei Arten von Programmen. Zum einen sind da die HAUPTSPEICHER-Programme, diese befinden sich irgendwo in den ersten 64 KB (E-RAM Konfig. &7FC0). Diese Hauptspeicher-Programme können maximal 40 KB lang sein, wenn sie bei &0000 beginnen und bei &9FFF enden. Notfalls kann ein Programm auch 44 KB erreichen und bis &AFFF reichen. Dabei werden allerdings die File-Tagging-Bytes (FTBs) überschrieben. Bytes oberhalb von &B800 niemals überschreiben (System Variablen!). Haufpspeicher-Programme enden mit der Extension .64K. Sie sind direkt mit dem RUN Icon startbar.

Beachte: Der RAM-Bereich von &0000 bis &3FFF wird von vielen Routinen benutzt, und zwischen &3800 und &3FFF muss ein Zeichensatz vorhanden sein. Schalte das untere ROM ein, um den ROM Zeichensatz zu nutzen.

* Zum anderen sind da die ERWEITERUNGS-SPEICHER-Programme. Ihre Größe hängt nur vom verfügbaren Speicher ab, maximal sind sie also 4 MB groß. Ihre Namen sollten mit der Extension .X16 enden.

Ein E(=Erweiterungsspeicher)-Programm hat einen speziellen Aufbau. Es wird normalerweise durch ein kurzes Programm geladen. Dieser Lader prüft, ob genug E-RAM frei ist, und wo das Programm hin soll. Das E-Programm befindet sich komplett im E-RAM. Die entsprechenden E-RAM Variablen XRAM_C4 ... XRAM_FF müssen angepasst werden, das wird von OS System Aufrufen erledigt (siehe später).

AKT_RAM muß vor dem Laden des E-RAM Programms mit dem korrekten Start E-RAM Block belegt werden z. B. &7FC4 für den ersten E-RAM Block.

```
LD BC,&7FC4
LD (AKT_RAM),BC ;wenn Prog. im Block &7FC4 (erstes E-RAM) beginnt.
```

Da das E-Programm in seperate 16 KB Blöcke gegliedert ist, ist nicht nur bei der Programmierung darauf zu achten, daß das Banking klappt.

An die Adresse &8000 sollte in etwa folgende Routine kopiert werden, damit eine automatische Blockumschaltung erfolgt:

```
ORG  &8000         ;Routine liegt auf &8000
PUSH AF            ;Register sichern
PUSH BC
LD   BC,(AKT_RAM) ;alter RAM-Status
INC  C
SET  2,C           ;nächster 16KB E-RAM Block selektieren
OUT  (C),C         ;neuen Block einblenden
LD   (AKT_RAM),BC ;und vermerken
POP  BC
POP  AF            ;Register restaurieren
JP   &4000         ;weiter gehts im neuen Block
```

Durch die Gliederung in 16 KB Blöcke existieren Einschränkungen, ein String z.B. muss komplett in einem 16 KB Block enthalten sein. Zu einem Zeitpunkt können nur Subroutinen eines Blocks benutzt werden.

Theoretisch können auch 64 KB Bänke umgeschalten werden, dabei hat man allerdings keinen Zugriff auf den Bildschirmspeicher.

Vordergrund- und Hintergrund- Programme

Unabhängig davon, ob ein Programm nun ein Hauptspeicher- oder Erweiterungsspeicher-Progrmm ist, kann dieses Programm auch ein Vordergrund- oder Hintergrund-Programm sein.

Was ist nun ein Vordergrund-Programm? Wird ein neues Programm geladen, dann wird es automatisch zum Vordergrund-Programm. Sein Header wird vom FutureOS nach Adresse &BC00 kopiert. Die 128 Bytes von &BC00 bis &BC7F enthalten den Datei-Header des zuletzt geladenen Programms (oder Datei).

Nach Programm-Ende kann das Vordergrund-Programm mittels des RUN Icons neu gestartet werden, ohne daß das OK Icon verwendet werden muß.

Läd oder startet man nun aber ein weiteres Vordergrund-Programm, so ist der Header des alten Vordergrund-Programms verloren. Damit ist auch das Programm verloren.

Um nun Programmen die Möglichkeit zu geben sich dauerhaft im RAM des CPC zu verankern kann es zum Hintergrund-Programm werden.

Hintergrund- und Vordergrund-Programme sind mit einem Unterschied gleich: Der Header eines Hinergrund-Programms beginnt ab &BC80. Das Programm muss seinen eigenen &80 Byte Header von &BC00 nach &BC80 kopieren.

Bevor sich ein Programm nun aber selbst zum Hintergrund-Programm macht, muss es testen ob nicht schon ein Hintergrund-Programm vorhanden ist (durch simples testen der Header-Prüfsumme). Ist dies der Fall, dann muss der Anwendner das alte Hintergrund-Programm beenden. Jedes Hintergrund-Programm muß die Option haben, sich zu deaktivieren, d. h. seinen Header ab &BC80 zu löschen.

Alternativ ist es möglich den Header des alten Hintergrundprogramms auf einen Header-Stapel (Stack) zu legen. Nach dem Ende des neuen Hintergrund-Programms wird der alter Header zurück installiert. Auf diese Weise ist ein Shell-Stack realisierbar.

Interrupts

Das FutureOS arbeitet im Interrupt Modus 2 (Vektor Interrupt). Dieser Interrupt-Modus ist sehr mächtig, denn ein externer Interrupt kann zu einer beliebigen Adresse verzweigen. So kann jeder Anwender seiner eigenen Hardware auch problemlos seine eigenen Routinen zuweisen. Die Interrupts sind beim Start von FutureOS ausgeschalten, können aber jederzeit aktiviert werden.

Will ein Programm im Interrupt Mode 1 arbeiten, so kann es sich seine eigenen Interrupt Routine ab &0038 installieren. Dazu muß es natürlich zuvor in den IM 1 umschalten. Außerdem ist darauf zu achten, das das untere RAM eingeschalten ist, und ein Zeichensatz ab &3800 im RAM zur Verfügung steht. Im IM 1 muss das untere ROM ausgeschalten sein. Vor dem Rücksprung ins Desktop muß der Vektor-Interrupt selektiert werden und die Interrupts sollten ausgeschalten werden (`DI:IM 2`). Abgesehen davon muß das I Register erhalten bleiben, es hat normalerweise den Wert &BD (`LD A,&BD:LD I,A`). Dies entspricht dem High-Byte der Vektor-Interrupt Sprungtabelle.

Der NMI (ab Adresse &0066) steht allen Programmen zur Verfügung, die ihn nutzen wollen. z.B. Inicron Netzwerk.

Das Desktop

Nach dem Start von FutureOS mit dem Befehl !OS oder !FDESK gelangt man in die Turbo-Desktop Benutzeroberfläche. Das Bild ist dreigeteilt. In der oberen Hälfte des Bildschirms befinden sich Icons, die der Auswahl von Speichermedien oder Funktionen dienen. Darunter befindet sich das Datei-Fenster, in dem pro Seite bis zu 64 Dateinamen angezeigt werden.

Mit den Tasten Shift und Control kann hier seitenweise auf und ab geblättert werden. Die untersten beiden Zeilen im Bild fungieren als Statuszeilen. Die Bedienung erfolgt mit Hotkeys, Joystick, Cursor-Tasten und Copy, Maus, Lichtgriffel, Grafpad II, Analog-Joystick etc.

Alle Menues, die durch anklicken eines Icons oder Hotkeys aufgerufenen werden, können jederzeit durch drücken von ESC verlassen werden.

Bei der Eingabe von Zahlen (hexadezimal oder dezimal) kann ebenfalls duch ESC (bzw. erst DEL, dann ESC) abgebrochen werden.

Menüpunkte können entweder durch den Druck der entsprechenden Zifferntasten, oder durch Auf- / Ab- Bewegung des Joysticks bzw. der Cursortasten angewählt werden. Mit Feuer oder Copy wird bestätigt.

Die Icons der Speichermedien A...M:

Die dunkel dargestellten Icons A...M können durch anklicken selektiert werden, damit wird das entsprechende Medium ausgewählt. Ein zweiter Klick deselektiert das Medium wieder. Schraffierte Icons sollten nicht benutzt werden, denn die entsprechenden Medien sind nicht verfügbar. Um ein Gerät auszuwählen kann man auch Space und dann A bis M drücken.

Das DIR Icon (Hotkey D):

Anklicken dieses Icons (Hotkey: D) liest die Inhaltsverzeichnisse von allen markierten Speicher-Medien ein. Anschließend wird die erste Seite des ersten Inhaltsverzeichnisses angezeigt. Mit den Tasten SHIFT und CONTROL lassen sich die DIRs auf und ab blättern.

Um Dateioperationen auszuführen, müssen die Inhaltsverzeichnisse der entsprechende Speichermedien zuerst eingelesen werden, denn unter FutureOS werden alle Inhaltsverzeichnisse im E-RAM gepuffert.

Versucht man das Inhaltsverzeichnis einer nicht eingelegten Diskette zu lesen, so wartet das OS etwa fünf Sekunden bis zur Sperrung des Laufwerks. In dieser Zeit kann die Disk eingelegt werden. An sonsten wird das LW gesperrt und sein Icon wird schraffiert dargestellt. In diesem Falle bitte Disk einlegen und das DIR Icon nochmal anklicken.

Wird in der OS Konfiguration (z.B. mittels 'Konfig OS') die Auto-DIR Funktion aktiviert, so werden alle Verzeichnisse (DIRs) automatisch nach dem Anklicken eines Speichermediums (A bis M) eingelesen.

Das Markieren von Dateien:
Wird eine Seite eines Inhaltsverzeichnisses gezeigt, so können die dargestellten Dateinamen mit dem Mauspfeil durch anklicken markiert werden. Markierte Dateien werden unterstrichen dargestellt. Klickt man eine markierte Datei ein zweites Mal an, so wird die Markierung wieder entfernt. Unter FutureOS beziehen sich Dateioperationen ausschließlich auf markierte Dateien.

Das TYPE Icon (Hotkey V):
Um Datei(en) auf dem Bildschirm zu zeigen markiert man zuerst die gewünschte(n) Datei(en). Dann aktiviert man das TYPE Icon (Hotkey: V). Um Texte oder Bilde anzuzeigen bitte die Taste "1" drücken. Zuerst wird der Datei-Header (falls vorhanden) angezeigt. Nach Tastendruck wird nun entweder das Bild angezeigt, oder es erscheint ein weiteres Menu, um das Bildschirmformat (Spalten und Zeilen) zu wählen. Die "1" zeigt den Text im gewohnten Format von 80 Zeichen auf 25 Zeilen. Nachdem das OS die Text-Datei geladen hat, wird die erste Seite angezeigt. Mit den Cursortasten oder dem Joystick kann auf und ab geblättert werden. Zum Beenden kann COPY oder ESC gedrückt werden. Oder man scrollt bis zum Ende der Datei. Wird ein Bild angezeigt, so können MODE und Format mittels der Cursor-Tasten eingestellt werden. Falls Sie mehr als eine Datei markiert haben, wiederholt sich der Vorgang ab dem zeigen des Headers für jede Datei.

Menuepunkt "2" des TYPE Icons dient dazu den 128 Byte Header einer markierten Datei anzuzeigen. Nach der Anzeige des ersten Headers bitte Space drücken um den Header der nächsten markierten Datei anzuzeigen. ESC bricht den Vorgang ab. Die letzte Datei kann im Desktop sofort mit dem RUN Icon gestartet werden (da sie noch markiert ist).

Das LOAD Icon (Hotkey L):
Um die erste markierte Datei zu laden wird das LOAD Icon geklickt. Im nun gezeigten Menue wird über die Ladeart entschieden.

Ladeart 0: Die Datei wird entsprechend ihres Headers geladen. Das Ziel kann dabei irgendwo im RAM oder Erweiterungs-RAM liegen.

Ladeart 1: Läd eine Datei in die ersten 64 KB Hauptspeicher an Adresse &0000. Solche Dateien enden oft mit der Extension "64K".

Ladeart 2: Läd eine Datein in das Erweiterungs RAM. Die Start-Adresse ist &4000 im ersten 16 KB Erweiterungs-RAM Block (&7FC4). Die Datei kann dabei bis zu 4 MB lang sein. Solche Dateien nutzen oft die Extension "X16".

Die Ladearten "3" und "4" laden Dateien an jede beliebige Adresse in den ersten 64 KB RAM (3) oder in die bis zu 4 MB Erweiterngs-RAM (4).
- Alle Werte müssen in hexadezimalen Zahlen angegeben werden.
- Bei der E-RAM Selektion müssen physikalische Werte angegeben werden.

Das SAVE Icon (Hotkey: S):

Zuvor geladene Dateien oder beliebige Speicherbereiche können mit SAVE gesichert werden. Es kann nur auf aktivierte Medien gesichert werden, deren Inhaltsverzeichnis zuvor mit DIR gelesen wurde. Vier Arten:

1. Vordergrund-Programm sichern: Sichert ein zuvor gestartetes oder geladenes Programm wieder auf ein Speichermedium. Laufwerk, User-Nummer, Dateiname und Extension können editiert werden.
Das Ziellaufwerk ist durch die Buchstaben A bis M symbolisiert.

2 Hintergrund-Programm sichern: Diese Funktion entspricht Punkt 1, es wird jedoch das gerade aktive Hintergrund-Programm gesichert.
Dazu muss zuvor ein Hintergrund-Programm gelanden worden sein.

3 Hauptspeicher sichern: Hiermit kann eine Datei aus einem definierten 64 KB RAM Block gesichert werden.
Durch das Editieren des Strings "A00:ProgNameExt" werden Medium, User-Nummer, Datei-Name und -Extension angegeben. Die Startadresse wird in hexadezimalen Zahlen angegeben. Will man aus den ersten 64 KB des Hauptspeichers sichern, dann ist die RAM-Konfiguration bei der Frage nach dem "Quell-Block" als &C0 anzugeben. Die Datei-Länge wird ebenfalls in hexadezimal abgefragt, sie ist in KB anzugeben.
Die Dateilänge sollte 64 KB nicht überschreiten.

4 Erweiterungs-RAM sichern: Diese Funktion dient dazu einen Datei aus dem Erweiterungs-RAM zu sichern. Bei der Frage nach der RAM-Konfiguration kann eine beliebige physikalische RAM Konfiguration angegeben werden. Die Dateilänge hängt nur vom E-RAM ab.
- Alle Werte müssen in hexadezimalen Zahlen angegeben werden.
- Bei der E-RAM Selektion müssen physikalische Werte angegeben werden.
- Die Dateilänge wird in KB angegeben. &0010 bedeutet z.B. 16 KB.

Das OK Icon (Hotkey O):

Es bietet die Möglichkeit in eine zuvor gestartete Applikation zurück zu kehren, falls diese Funktion vom Programm unterstützt wird.

Ein Applikation kann ins Desktop springen, dort können z.B. Dateien markiert werden (etc.). Anschießend wird das OK Icon geklickt, dadurch wird die Applikation wieder aufgerufen.

Klickt man das OK an, ohne dass es von einer Applikation verwendet wird, so erscheint eine Fehlermeldung.

Das <I> Icon (Hotkey I):

Es bietet die Möglichkeit ein zuvor gestartetes Informations-System aufzurufen. Das I Icon arbeitet nach dem selben Prinzip wie das OK Icon. Wurde zuvor kein solches System initialisiert, so wird mit einer Fehlermeldung abgebrochen.

Das ERAse Icon (Hotkey E):

1) Datei(en) löschen: dieser Punkt löscht die markierten Dateien eines oder mehrerer Laufwerke. Man kann alle Dateien mit Punkt "4) Alle & sofort" löschen. Die Dateien sind dann garantiert gelöscht! Oder man wählt Punkt "5) mit Sicherheitsabfrage" - dabei wird für jede Datei nochmals gefragt, ob sie wirklich gelöscht werden soll.

2) Disk formatieren: mit diesem Punkt formatiert man Disketten, und zwar im Data-, System-, IBM- oder Vortex Format. Nach der Formatwahl folgt dann die Frage nach dem zu formatierenden Laufwerk. Bitte zuerst die Diskette einlegen, und dann erst das Laufwerk (mit den Tasten A bis H) selektieren.

Das RENname Icon (Hotkey N):

Unter Punkt 1) ist es möglich Datei(en) umzubenennen. Zuerst erscheint der Name der ersten markierten Datei in doppelter Ausführung. Der untere Dateiname kann nun geändert werden, auch die Usernummer kann editiert werden. Darüber ist der alte Dateiname dargestellt. Keinesfalls darf die Usernummer auf &FF gesetzt werden. Bei CP/M Disks mit Time-Stamps darf kein RENname durchgeführt werden.

Das COPY Icon (Hotkeys C oder F):

Punkt 1) bietet die Möglichkeit Dateien multidirektional zu kopieren. D. h. Dateien können in einer Operation von mehreren Quelle an unterschiedliche Ziele kopiert werden.

Kopiert werden alle markierten Dateien. Im Untermenu entscheidet man, ob alle Dateien direkt in ein Ziel kopiert werden sollen (Punkt 3.) oder ob man Ziel-Medium, User, Name und Extension einer jeden Datei editieren will.

Punkt 2) dient dem Kopieren von Disketten. Es kann mit Data-, System, IBM- und Vortex Format gearbeitet werden. Die Verwendung beider Disk-Seiten und von DoubleStep ist möglich. Sind Quell- und Ziel-Laufwerk identisch, so müssen Quell- bzw. Ziel- Disk nach Aufforderung eingelegt werden. Das Format der Disks wird selbstständig ermittelt, bei Bedarf wird die Ziel-Disk automatisch formatiert.

Das DRUCKEN Icon (Hotkey P):

Sowohl Inhaltsverzeichnisse (Punkt 1) als auch markierte Dateien (Punkt 2) lassen sich ausdrucken.

Das ReTag Icon (Hotkey A):

Es dient dazu alle bereits bearbeiteten Dateien (durchgestrichen dargestellt) wieder neu zu markieren (unterstrichen). Dadurch ist es möglich eine bestimmte Auswahl von Dateien mehrmals zu bearbeiten.

Das UnTag Icon (Hotkey U):

Seine Funktion ist die ENTmarkierung aller Dateien. Der selbe Effekt läßt sich auch erzielen indem man das DIR Icon anklickt, dann werden allerdings alle DIRs neu gelesen.

Das MONitor Icon (Hotkey M):

Es dient dem Einsprung in den Maschinesprache-Monitor. Weitere Infos, siehe folgendes Kapittel.

Das WECKER Icon (Hotkey W):

Damit läßt sich die Weckzeit einstellen, Cursortasten selektieren die Ziffer, und deren Wert. Copy beendet das Editieren ordnungsgemäß. ESC bricht ohne Änderungen ab.

Das END Icon (Hotkey Q):

Es dient dazu FutureOS zu beenden. Man hat auch die Möglichkeit (Punkt 1) FutureOS neu starten. Will man zurück ins BASIC, dann wählt man Punkt 2) AMS-OS initialisieren.

Das RUN Icon (Hotkey X):

Wenn beim Anklicken des Icons irgendeine Datei markiert ist, dann wird sie in den 64 KB Haupt-Speicher geladen und dort ausgeführt. Dabei wird immer die erste markierte Datei verwendet. Gestartet wird ab der Autostartadresse bzw. ab der Ladeadresse der Datei, wenn der Autostart &0000 ist.

Ist keine Datei markiert, dann hat man die Wahl das aktuelle Vordergrund Programm (Punkt 1) oder das aktuelle Hintergrund Programm (Punkt 2) zu starten.

Unter Punkt 3) besteht die Möglichkeit in die ersten 576 KB RAM einzuspringen. Vorsicht man sollte wissen wohin.

Punkt 4) ermöglicht es eine Applikation in einem Erweiterungs-ROM für FutureOS aufzurufen. Dabei selektiert man zuerst das XROM, dannach die Applikation.

Dabei sollten die XROMs aufeinanderfolgende ROM Nummern besitzen, um mit "Auf" und "Ab" das gewünschte XROM selektieren zu können.

Das IDE Icon (Hotkey H):

Das IDE Icon dient dazu das IDE ROM zu aktivieren, um mit IDE Geräten arbeiten zu können. Das sind z.B. X-MASS, Jareks IDE8255, CPC-IDE, Albireo oder SYMBiFACE II. Dafür ist ein fünftes ROM (IDE-ROM) nötig.

Die Uhrzeit- & Datums- Icons (Hotkeys Y and Z):

Sie befinden sich links(Zeit) bzw. rechts(Datum) in der untersten Icon-Reihe. Damit lassen sich Uhrzeit bzw. Datum einstellen. Dies geschieht mit den Cursortasten. Copy beendet. ESC bricht ab.

Zusätzliche Hotkeys

- Mit "j" (jump) kann man auf den Namen der ersten Datei in Dateifenster springen.

- Mit "b" (bypass) kann man den Mauspfeil auf den Namen der nächsten Datei verschieben, ohne die aktuelle Datei zu markieren.

Auf diese Weise lassen sich mit G und B alle Dateien markieren bzw. entmarkieren und anschließend bearbeiten.

V. Der Maschinen-Monitor

Der Maschinen-Monitor stellt ein mächtiges Werkzeug dar. Mit seiner Hilfe ist es möglich jede Speicherstelle des Systems gezielt zu verändern, 16 Bit I/O Adressen anzusprechen, damit die Hardware direkt zu programmieren und Routinen zu starten. Wenn man eine Routine starten oder testen will kann man das RUN icon oder den Monitor nutzen.

Der Monitor erlaubt es alles Z80 Register vor dem Aufruf einer Routine zu setzen. Aber bitte nutzen Sie den Monitor nur, wenn sie sich mit der Hardware des CPC gut auskennen. An wichtigen Stellen ist der Abbruch mit ESC möglich. Alle Zahlen-Werte bitte im Hexadezimalsystem eingeben.

d = FutureOS - <D>ump/Edit System

Diese Funktion zeigt einen Teil des Speichers hexadezimal und in ASCII Zeichen. Zuerst wird man nach einer Start-Adresse gefragt. Nach deren Eingabe werden &01E0 Bytes dargestellt. Pro Zeile werden je 16 Bytes gezeigt, links als Hex-Werte, rechts als ASCII Zeichen. Durch die Cursor Tasten AUF und AB kann man um je eine Seite nach oben oder unten blättern.

Drückt man "e", dann kommt man in den Editor-Modus. Jede andere Taste führt ins Menü des Monitors. Im Editormodus wird zuerst links oben ein Cursor dargestellt. Er kann durch alle vier Pfeiltasten bewegt werden. Will man ein Byte verändern, dann drückt man zuerst die "COPY" Taste, dann ist der Wert hexadezimal einzutippen, die Eingabe wird mit Return beendet oder mit ESC abgebrochen. Um den Editor-Modus zu verlassen ist das kleine Enter zu drücken.

Der mit "k = Konfig" eingestellte RAM / ROM / MM Bereich wird von Dump / Edit System beachtet.

p = FutureOS - <P>ort Ein-/Ausgabe

Diese Funktion dient dazu Bytes an eine Portadresse zu schicken oder zu lesen.

Der Cursor ist mit den Pfeiltasten zu bewegen. ESC führt ins Menü des Monitors zurück.

Sollen Daten von einer I/O Adresse gelesen werden, ist der Cursor über die entsprechene Adresse zu führen ("EING" Spalte) und COPY zu drücken um ein Byte zu lesen.

Um ein Byte an eine I/O Adresse zu schicken bewegt man den Cursor in die "AUSG." Spalte. Dort wird nach dem Druck auf "COPY" ein 8-Bit Wert (hexadezimal) erfragt. Das Byte wird durch drücken von RETURN an die I/O Adresse gesendet. ESC bricht die Eingabe ab.

r = Z80 <R>egister editieren

Diese Funktion erlaubt es allen Z80 Register Werte zuzuweisen. Wird eine Routine vom Monitor aus aufgerufen, so wird die Z80 zuvor mit den hier eingegebenen Werten geladen.

Alle Prozessorregister werden mit ihren Inhalten gezeigt. Mit den Cursortasten kann ein Register selektiert werden, COPY erlaubt die Eingabe eines hexadezimalen Wertes, ESC bricht ab. Das kleine Enter führt ins Menue zurück. Die hier gezeigten Werte werden in den 16 Bit RAM-Variablen REG_AF bis REG_PC gespeichert.

Springt eine Routine zurück in den Maschinen Monitor so werden diese RAM Register mit den Werten der Z80 Register geladen.

Springt eine Routine die OS Funktion CARET an, so werden alle Register in diese Variablen gesichert und man findet sich im Hauptmenue des Monitors. CARET dient der Analyse der Aussprung-Bedingungen einer Maschinensprache-Routine.

k = RAM/ROM/MM <K>onfiguration

Dieser Punkt definiert ob mit RAM oder ROM gearbeitet wird und welche RAM Konfiguration genutzt wird. Alle anderen Punkte des Monitors halten sich an diese Einstellungen.
Zuerst wird man gefragt ob das untere ROM (enthält den Zeichensatz!) eingeblendet werden soll. Man kann "j" für Ja oder "n" für Nein tippen. Wird Nein gewählt, so wird das untere RAM eingeblendet.
Nun wird man nach der physikalischen RAM-Konfiguration gefragt. Man sollte jedoch nur die Konfigurationen &C0 (1. 64 KB ein) bzw. &C4, ...,&FF wählen (16 KB E-RAM aus den ersten 32 Ext-RAMs, zwischen &4000 und &7FFF eingeblendet).
Beim CPC Plus wird zusätzlich abgefragt, ob die Memory-Mapped Baugruppen aktiviert werden sollen. Man tippe wieder "j" für Ja oder "n" für Nein. Aktive MM-Baugruppen werden im Bereich zwischen &4000 und &7FFF eingeblendet. Die MM-Baugruppen haben die höchste Priorität.

v = Speicherbereich <V>erschieben

Dieser Punkt dient dazu einen Speicherbereich beliebiger Länge an eine beliebige Adresse im RAM zu kopieren.
Zuerst wird man nach der Start-Adresse gefragt. Damit ist das erste Byte des zu verschiebenden Blockes gemeint.
Nun folgt die Frage nach der End-Adresse, damit ist das letzte noch zu verschiebende Byte gemeint.
Nachdem man als drittes die Zieladresse eingegeben hat, wird der Block an die Ziel-Adresse kopiert.
Achtung! Der Bereich von &B800 bis &BFFF darf NICHT(!) überschreiben werden. Hier liegen das OS System RAM und die OS System Variablen.

i = Speicherbereich <i>nitialisieren

Diese Funktion dient zur Initialisierung eines Speicherbereichs. Nachdem man Anfangs- (erstes Byte des Blocks) und End-Adresse (letztes Byte des Blocks, inclusive) eingegeben hat, wird man nach einem 16 Bit Wert gefragt, mit dem der definierte Speicherbereich gefüllt werden soll. Die Initialisierung erfolgt anschließend.

a = <A>ufruf Routine

Mit diesem Punkt lassen sich Routinen aufzufufen, dabei werden die Z80 Register aus den (zuvor editierten) RAM-Variablen geladen (s.o.).

Man wird nach der Start-Adresse gefragt, ab der die aufzurufende Routine beginnt. Nach Eingabe der Adresse drückt man RETURN um die Routine aufzurufen bzw. man drückt (DEL und) ESC um diesem Punkt abzubrechen.

Die RAM und ROM Konfiguration wird dabei so gewählt, wie zuvor unter Punkt "K"onfiguration gewählt.

x = FutureOS - MONITOR verlassen, e<X>it

Diese Funktion dient dazu den Monitor zu verlassen und ins Desktop zurückzukehren. Der Monitor kann nicht durch ungewolltes Drücken von ESC verlassen werden.

VI. Genauere Erklärung der einzelnen Fähigkeiten des OS

Die Tastaturverwaltung

Die Verwaltung der Tastatur ist unter FutureOS in vier Ebenen organisiert. Die Tastatur wird nur bei Bedarf abgefragt. Dabei werden die beiden Digital-Joysticks als Bestandteil der Tastatur behandelt.

Wärend BASIC die Tastatur 50 mal pro Sekunde abfragt, findet die Abfrage unter FutureOS nur bei Bedarf statt. Diese Vorgehensweise spart etwas Rechenzeit. Will man den Status einiger oder aller Tasten ermitteln, dann ruft man die entsprechende OS Funktion auf, die den aktuellen Tastatur-Status liefert. Später wird gezeigt, daß die Abfragemöglichkeiten sehr vielfältig sind.

Unter Basic existieren drei Tastaturebenen, bei FutureOS kommt eine vierte hinzu. Es ist eine NORMAL, eine SHIFT, eine CONTROL und eine kombinierte SHIFT+CONTROL Ebene vorhanden.

Einschränkend können bei der kombinierten SHIFT + CONTROL Ebene einige Tasten nicht verwendet werden. Die Hardware-Organisation der Tastaturmatrix ist dafür verantwortlich. Die folgenden neun Tasten können nicht benutzt werde:

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

Bei Anschluss einer externen Tastatur können diese Tasten oft dennoch verwendet werden.

Worin liegt der Vorteil einer vierten Tastatur-Ebene? Nun, die vierte Ebene erlaubt es alle 256 möglichen Zeichen auf die 80 Tasten der Tastatur zu verteilen. Hätte man nur drei Ebenen zur Verfügung, dann wären nur 3 * 80 = 240 Zeichen darstellbar, nicht aber alle 256 möglichen 8 Bit Zeichen des Zeichensatzes.

Die Tastaturbelegung:

Für jede der vier Tastaturebenen ist eine 80 Byte lange Tabelle im System RAM des OS reserviert, diese Tabellen entsprechen jeweils der Tastaturmatrix.

Durch die Änderung eines Bytes dieser Tabellen ist es möglich einer bestimmten Taste einen bestimmten Wert zuzuweisen.

```
Die Tabelle der NORMAL ------ Ebene steht im RAM ab TAST_N  = &B980
Die Tabelle der SHIFT ------- Ebene steht im RAM ab TAST_S  = &BA00
Die Tabelle der CONTROL ----- Ebene steht im RAM ab TAST_C  = &BA80
Die Tabelle der SHIFT+CONTROL Ebene steht im RAM ab TAST_SC = &BB00
```

Eine ROM Kopie dieser vier Tabellen befindet sich im C-ROM ab Adresse XAST_N = &FDAA. Wenn dort ein Byte verändert wird, dann ist diese Änderung für alle FutureOS Sitzungen gültig. Aber Vorsicht, verlieren Sie keine wichigen Tasten.

OS Funktionen der Tastaturverwaltung

Unter FutureOS stehen verschiedene OS Funktionen zur Tastaturabfrage zur Verfügung. Sie sollten sich zu diesem Thema unbedingt die Datei ‚**API-A-DE'.DOK** ansehen, denn dort sind die im folgenden besprochenen OS Funktionen ausführlich dokumentiert.
Eine nützliche OS Funktion stellt H_XALLET dar. Sie liefert das ASCII Zeichen das der aktuell gedrückten Taste entspricht. Dabei werden die SHIFT, CONTROL und CAPS Tasten beachtet.
Daneben stehen weitere OS Funktionen zur Verfügung:

Wichtige OS Funktionen zur Tastatur-Verwaltung

H_XALLET	fragt alle Tasten ab; Caps-Lock, Shift, Control wird beachtet
HOLE1TS	testet ob eine bestimmte Taste gedrückt ist
H_JC	fragt beide Joysticks, die Cursortasten und das MultiPlay ab. H_JC ist Bit-kompatibel zu H_CCE und H_JOY
M_CON	fragt proportionale Mäuse ab (SYMBiFACE 2, SF3, MultiPlay) Das Ergebnis in A ist Bit-kompatibel zu OS Funktion H_JC
H_JCM	eine Kombination aus H_JC und M_CON. Testet alle HID Geräte
XWART	wartet variable Wartezeit, Anfangs- und Folgeverzögerung
WATA	wartet bis irgendeine Taste gedrückt wurde
RB_8	8 Bit Eingabe, springt je nach Zahlensystem(hex/dez) zu BIT8_IN oder B8DIN
RB_16	16 Bit Eingabe, springt je nach Zahlensystem (hex/dez) zu B16IN oder B16DIN
STED	zeigt und editiert einen String (16 Bit String-Laenge)

Weitere OS Funktionen zur Tastatur-Verwaltung

H_ALLET	fragt alle Tasten ab, ohne Caps-Lock o.ä. zu beachten
TST_TS	testet ob überhaupt irgendeine Taste gedrückt ist (Ja/Nein)
WART_TS	wartet bis garantiert keine Taste mehr gedrückt ist
HOLETST	liest den kompletten Status aller 80 Tasten ins RAM
A_F_0_9	liest einen Wert von 0-9 oder a-f von Tastatur ein und stellt diesen auf dem Bildschirm (Mode 2) dar
H_CURA	fragt nur die vier Cursor- und die Copy- Taste ab
H_CCE	fragt Cursortasten, Copy und kleines Enter ab, Joy-kompatibel
CUR_CPY	fragt Cursortasten, Copy und die ESC-Taste ab
H_JOY	fragt den Status beider Digital-Joysticks ab
H_CS	fragt die Tasten SHIFT und CONTROL ab
BIT8_IN	stellt eine Eingabefunktion für 8-Bit Werte dar, hexadezimal
B8DIN	stellt eine Eingabefunktion für 8-Bit Werte dar, dezimal
B16IN	stellt eine Eingabefunktion für 16-Bit Werte dar, hexadezimal
B16DIN	stellt eine Eingabefunktion für 16-Bit Werte dar, dezimal

Für die durchschnittliche Applikation sind die OS Funktionen H_JCM oder H_JC, H_XALLET, XWART, WATA und STED ausreichend.

Die Tastaturmatrix der CPCs

	b7	b6	b5	b4	b3	b2	b1	b0
z0:	.(f)	ENTER	f3	f6	f9	CurU.	CurR.	CurO.
z1:	f0	f2	f1	f5	f8	f7	Copy	CurL.
z2:	CTRL	\	SHIFT	f4	]	RET	[	CLR
z3:	.	/	:	;	P	@	-	^
z4:	,	M	K	L	I	O	9	0
z5:	SPACE	N	J	H	Y	U	7	8
z6:	V	B	F	G	T	R	5	6
z7:	X	C	D	S	W	E	3	4
z8:	Z	Caps	A	TAB	Q	ESC	2	1
z9:	DEL	Feu.3	Feu.2	Feu.1	JoyR.	JoyL.	JoyU.	JoyO.

Dabei bedeutet JoyR. = rechts, L = links, U = unten und O = oben.
.(f) ist der Punkt im Block der f-Tasten. z0 bis z9 sind die Zeilen der Tastatur-Matrix.

Zeichen und Strings

Unter BASIC existiert ein zentraler Einsprung um Zeichen auszugeben. Diese Zeichenausgabe ist sehr mächtig, funktioniert in allen Modis, erlaubt verschiedene Print-Modi usw. Der Preis dieser Vielfalt ist die relativ langsame Geschwindigkeit.

FutureOS bietet unterschiedliche Möglichkeiten Zeichen, Strings oder Terms auf dem Bildschirm darzustellen. Terms sind Strings die Control-Codes enthalten. Zeichen können in Mode 1 und 2 dargestellt werden. Diese Spezialisierung macht die Zeichenausgabe des FutureOS bis zu 80 mal schneller als BASIC.

Achtung: Für jegliche Art von Zeichenausgabe ist ein Zeichensatz nötig. Dieser muß sich im Bereich von &3800 bis &3FFF befinden. Entweder man blendet das untere ROM (lower ROM) ein, oder man läd einen Zeichensatz ins RAM. Siehe RAM-Variable: RAMCHAR (&B847).

Die Steuerzeichen des Future Operating System:

Die Steuerzeichen des FutureOS sind in allen Funktionen völlig frei

definierbar. Allerdings werden nur die ersten 32 Zeichen als Steuerzeichen interpretiert (0-31 / &00-&1F). Die ersten 32 Zeichen werden nur als Steuerzeichen interpretiert, wenn eine Zeichenkette mit einer der Terminal-Funktionen ausgegeben wird z.B.: TERM_2, TER_BB ... Welche Steuerzeichen von FutureOS wie genutzt werden wird im folgenden detailiert erläutert.

Will man z.B. ein fremdes Terminal emulieren, oder einfach seine eigenen Funktionen in die Textausgabe integrieren, so lassen sich dafür alle 32 Steuerzeichen problemlos umdefinieren. Für Mode 1 und Mode 2 existiert jeweils eine Tabelle für alle 32 Steuerzeichen. Ein Steuerzeichen kann also je nach Mode eine andere Funktion ausführen. Jede dieser Tabellen besteht aus 32 Variablen (16 Bit). Jede Variable enthält die Adresse einer Routine, die der entsprechende Control Code aufruft. Die Tabellen sind jeweils 32 * 2 Byte = 64 Byte lang.

Steuerzeichen-Tabelle für Mode 1 reicht von: TAS_S1 = &B800 bis &B83F
Steuerzeichen-Tabelle für Mode 2 reicht von: TAS_S2 = &B900 bis &B93F

Die Tabellen beginnen je mit Steuerzeichen &00 (0), und enden mit Code &1F (31). Jede 16 Bit Variable enthält zuerst das Low-Byte und dann das High-Byte der Control-Code-Routine.

Funktionen der Steuerzeichen in Mode 1:

Im folgenden werden die in Mode 1 gültigen Steuerzeichen beschrieben, dabei existieren Unterschiede zu einigen Mode 2 Steuerzeichen.

Steuerzeichen, die in beiden Modes eine identische Funktion erfüllen, sind nur unter 'Funktionen der Steuerzeichen in Mode 2:' beschrieben.

Code &00 = 00 : Beendet die Ausgabe des Strings, siehe Mode 2.

Code &01 = 01 : Selektiert RAM-Zeichensatz (&3800-&3FFF), L-RAM ein.

Code &02 = 02 : Selektiert ROM-Zeichensatz (&3800-&3FFF), L-ROM ein.

Code &03 = 03 : Blendet E-RAM (&4000-&7FFF) ein, siehe Mode 2.

Code &06 = 06 : Halbes Space, die Cursor-Position wird um ein halbes Leerzeichen nach rechts bewegt. Dies ermöglicht halben Zeichenabstand.

Code &08 = 08 : x mal Space ausgeben, dem Code &08 folgt ein Byte, das angibt wieviele Spaces zur aktuellen Cursorposition addiert werden sollen. Es sind Werte zwischen 1 und 127 möglich.

Code &09 = 09 : TAB (1-8 Zeichen), siehe Mode 2.

Code &0A = 10 : LINEFEED ohne Return, siehe Mode 2.

Code &0B = 11 : Der Bildschirm wird gelöscht, entspricht CLS.

Code &0C = 12 : Cursor home, links oben, siehe Mode 2.

Code &0D = 13 : RETURN ohne Zeilen-Vorschub, siehe Mode 2.

Code &0E = 14 : Echtes RETURN, entspricht Codes &0A+&0D, siehe Mode 2.

Code &0F = 15 : Neue Text-Adresse setzen, siehe Mode 2.

Code &10 = 16 : Zeichenausgabe wird auf Pen 1 umgestellt. Alle Zeichen nach diesen Steuercode werden mit Pen 1 dargestellt.

Code &11 = 17 : Zeichenausgabe wird auf Pen 2 umgestellt.

Code &12 = 18 : Zeichenausgabe wird auf Pen 3 umgestellt.

Code &13 = 19 : Zeichenausgabe wird auf die Pen 1 und 2 (Mischfarbe) umgestellt. Dies ist die schnellste Zeichenausgabe für Mode 1.

Code &1A = 26 : Wie Code 0. Beendet die Stringausgabe, siehe Mode 2.

Code &1D = 29 : Einige Zeichen 8-fach zoomen, siehe Mode 2.

Code &1E = 30 : LOCATE 32 * 32, der Cursor kann an eine beliebige Position auf dem Bildschirm gesetzt werden. Dem Code &1E folgen zwei Bytes, das erste gibt die Y-Position (0..31) an, das zweite Byte gibt die X-Position (0..31) an. Dieser Steuercode ist nur im 32 * 32 Modus gültig, also wenn der Bildschirm z.B. mit OS Funktion S64X32 auf 32 Mode 1 Zeichen pro Zeile und 32 Zeilen gesetzt wurde.

Code &1F = 31 : LOCATE 40 * 25, der Cursor kann an eine beliebige Position auf dem Bildschirm gesetzt werden. Dem Code &1F folgen zwei Bytes, das erste gibt die Y-Position (0..24) an, das zweite Byte gibt die X-Position (0..39) an. Dieser Steuercode ist nur im 40 * 25 Modus gültig, also wenn der Bildschirm z.B. mit OS Funktion S80X25 auf 40 Mode 1 Zeichen pro Zeile und 25 Zeilen gesetzt wurde.

Funktionen der Steuerzeichen in Mode 2:

Code &00 = 00 : Beendet die Ausgabe des Strings auf dem Bildschirm. Jeder auszugebende Text muß mit einem Null-Byte abgeschlossen sein, sonst wird das Ende nicht erkannt. (&1A hat die selbe Funktion s.u.).

Code &01 = 01 : RAM-Zeichen (&3800-&3FFF) d.h. L-RAM einblenden.

Code &02 = 02 : Der ROM-Zeichensatz (unteres ROM) wird eingeblendet.

Code &03 = 03 : Blendet Erweiterungs-RAM zwischen &4000 und &7FFF ein. Code &03 selektiert einen 16 KB Block der ersten 512 KB E-RAM. Dadurch kann darin enthaltener Text ausgegeben werden. Dem Code &03 folgt ein Byte, dass das physikalische E-RAM selektiert: &C4, &C5, &C6, &C7, &CC, &D4 ... &FF. Byte &C0 selektiert die ersten 64 KB Hauptspeicher.

Code &04 = 04 : vertikal x mal y ausgeben, 80 * 25. Code &04 gibt ein Byte mehrmals vertikal untereinander aus. Dem Steuercode folgen zwei Bytes. Das erste Byte gibt an, wie oft das zweite Folgebyte ausgegeben werden soll. Dabei wird die Cursorposition lediglich um eins nach rechts geschoben. Code &04 wird im 80 (Zeichen) * 25 (Zeilen) Modus verwendet (siehe OS Funktion S80X25).

Code &05 = 05 : vertikal x mal y ausgeben, 64 * 32. Code &05 gibt ein Byte mehrmals vertikal untereinander aus. Dem Steuercode folgen zwei Bytes. Das erste Byte gibt an, wie oft das zweite Folgebyte ausgegeben werden soll. Dabei wird die Cursorposition lediglich um eins nach rechts geschoben. Code &05 ist für den 64 Zeichen * 32 Zeilen Modus vorgesehen (siehe OS Funktion S64X32).

Code &06 = 06 : rückt Cursor um eine Position nach rechts (SPACE).

Code &07 = 07 : horizontal x mal y ausgeben. Code &07 erlaubt es ein Zeichen beliebig oft auszugeben. Dem Code &07 folgen drei Bytes. Die ersten beiden geben an, wie oft das dritte Byte ausgegeben wird. Dabei ist Folgebyte 1 das Low-, und Folgebyte 2 das High-Byte der Anzahl.

Code &08 = 08 : x mal Space ausgeben, dem Code &08 folgt ein Byte, das angibt wie viele Spaces zur aktuellen Cursorposition addiert werden sollen. Es sind Werte von 1 bis 255 möglich.

Code &09 = 09 : TAB, die Cursorposition wird auf den nächsten Tabulator gesetzt. Die Tabulatoren existieren alle 8 Zeichen-Positionen.

Code &0A = 10 : LINEFEED, der Cursor wird um eine Zeile nach unten bewegt. Cursor wird nicht nach links gesetzt. Es erfolgt KEIN Return.

Code &0B = 11 : Der Bildschirm wird gelöscht, entspricht CLS.

Code &0C = 12 : Cursor home, die Cursorposition wird auf die linke, obere Ecke gesetzt. Weiterer Text wird ab dieser Position ausgegeben.

Code &0D = 13 : RETURN, der Cursor wird an den Anfang (links) der aktuellen Zeile gesetzt. Dabei erfolgt KEIN Zeilen-Vorschub.

Code &0E = 14 : Echtes RETURN, der Cursor wird an den Anfang (links) der nächstunteren Zeile bewegt. Der Code &0E entspricht &0A und &0D.

Code &0F = 15 : Neue Text-Adresse setzen, dem Code &0F folgen zwei Bytes (erst das Low-Byte, dann das High-Byte). Beide Bytes definieren eine 16 Bit Adresse, an der die Textausgabe fortgesetzt werden soll. Mit Code &0F kann von einem String in einen anderen gesprungen werden.

Code &10 = 16 : Zeichenausgabe wird auf "normal" umgestellt. Alle folgenden Zeichen werden ohen Attribute dargestellt.

Code &11 = 17 : Zeichenausgabe wird auf "invers" umgestellt.

Code &12 = 18 : Zeichenausgabe wird auf "kursiv" umgestellt.

Code &13 = 19 : Zeichenausgabe wird auf "unterstrichen" umgestellt.

Code &14 = 20 : Zeichenausgabe wird auf "durchgestrichen" umgestellt.

Code &1A = 26 : Beendet die Ausgabe des Strings auf dem Bildschirm. Er entspricht Code &00. Zwecks Kompatibilität zu ASCII vorhanden.

Code &1D = 29 : Einige Zeichen 8-fach gezoomt darstellen. Dem Code &1D folgt ein Byte, das angibt, wieviele der folgenden Textzeichen gezoomt dargestellt werden sollen. Dieser Wert sollte zwischen 1 und 8 liegen.

Code &1E = 30 : LOCATE 64 * 32, der Cursor kann an eine beliebige Position auf dem Bildschirm gesetzt werden. Dem Code &1E folgen zwei Bytes, das erste gibt die Y-Position (0..31) an, das zweite Byte gibt die X-Position (0..63) an. Dieser Steuercode wird im 64 * 32 Modus genutzt. Der Bildschirm kann mit OS Funktion S64X32 auf 64 Mode 2 Zeichen pro Zeile und 32 Zeilen pro Seite gesetzt wurde.

Code &1F = 31 : LOCATE 80 * 25, der Cursor kann an eine beliebige Position auf dem Bildschirm gesetzt werden. Dem Code &1F folgen zwei Bytes, das erste gibt die Y-Position (0..24) an, das zweite Byte gibt die X-Position (0..79) an. Dieser Steuercode wird nur im 80 * 25 Modus verwendet, d. h. der Bildschirm wurde zuvor mittels OS Funktion S80X25 auf 80 Mode 2 Zeichen pro Zeile und 25 Zeilen pro Seite gesetzt.

Neu- oder Um-Definition von Steuerzeichen:

Das FutureOS bietet die Möglichkeit einem Steuerzeichen eine beliebige Funktion zuzuweisen. Dies sollte man aber mit Vorsicht tun, da die Konventionen streng eingehalten werden müssen.

Wie kann man nun ein Steuerzeichen mit einer Funktion versehen oder einfach nur umdefinieren? Hier eine schrittweise Erklärung:

- Für jedes Steuerzeichen existieren zwei * zwei Bytes im RAM. Einmal zwei Bytes für Mode 1 und einmal zwei Bytes für Mode 2. Diese zwei Bytes beinhalten je eine 16 Bit Adresse (low -> high). Diese Adresse zeigt auf den Start der Routine des entsprechenden Steuerzeichens.

- Wird nun ein Steuerzeichen ausgeführt, so wird diese Adresse gelesen, und das dortige Unterprogramm ausgeführt (Mode abhängig).

- Um ein Steuerzeichen neu zu definieren, muß man lediglich die zugehörige Adresse in der Steuerzeichen-Tabelle ändern. Die Tabellen beginnen im System RAM ab TAS_S1 (Mode 1) und TAS_S2 (Mode 2). Die Tabellen beginnen mit der Adresse der Routine für Control Code &00.

- Nun ist das Steuerzeichen neu definiert.

Aber Achtung: Das Unterprogramm eines Steuerzeichens darf die ROM-Konfiguration NICHT verändern. Das ROM A muß eingeblendet bleiben. Eine Änderung der ROM Konfiguration gefährdet das System.

Weiterhin muß der Inhalt des Registers DE nach dem Abarbeiten der Steuerzeichen-Routine in Register HL enthalten sein. Wird DE nicht benutzt, dann genügt ein einfacher EX DE,HL. Wird auch DE benötigt: Siehe Listing unten...

Folgen einem Steuerzeichen einige Parameter-Bytes, so kann die entsprechende Routine diese macheinander ab der in DE enthaltenen Adresse lesen. Ansonnsten folgen ab DE die dem Steuercode folgenden Textbytes. Siehe Listing unten...

Beispiel:

Um dem Steuerzeichen &00 in Mode 2 eine neue Funktion zu geben muß man lediglich die Speicherstelle &B900 mit der Adresse laden, an der das Unterprogramm beginnt, welches die gewünschte Funktion ausführt. Ab &B900 beginnt die Steuerzeichentabelle für Mode 2. Will man das Steuerzeichen für Mode 1 umdefinieren, dann verwendet man Adresse &B800 anstatt &B900. Formel: SteuCode * 2 + &B800 oder &B900

Beispiel für ein Steuerzeichen:

```
STEUERZ PUSH DE       ; DE sichern
        ??????????    ; Unterprogramm des Steuerzeichens
        POP  HL       ; alten Wert von DE nun in HL holen
        RET           ; und zurück.
```

Beispiel: Control Code Subroutine um die Border-Farbe zu setzen:

```
SET_BOR LD  BC,&7F00
        OUT (C),C     ; BORDER Farbregister selektieren

        LD  A,(DE)
        INC DE        ; Farbwert (folgt dem Code) holen,

        OUT (C),A     ; und neue Border-Farbe setzen!

        EX  DE,HL     ; DE in HL transferieren
        RET           ; und zurück!
```

Will man die Steuerzeichen-Tabellen dem Zustand nach dem Hochfahren des OS anpassen, so verwendet man dazu OS Funktion CSTI aus ROM A.

Variablen der Zeichenausgebe

* Da wäre zuerst mal die Variable C_POS zu nennen, sie gibt die Cursor Position an. Sie enthält normalerweise einen Wert zwischen &BFFE und &FFFF. C_POS ist also ein Zeiger direkt ins Bildschrim- / Video-RAM.

Dieser Zeiger hat allerdings eine Mode-abhängige Abweichung. Wärend in Mode 2 C_POS um 1 kleiner ist als die reale Position, an der das nächste Zeichen ausgegeben wird, ist C_POS in Mode 1 um 2 kleiner. Dies hängt damit zusammen, daß die Zeichenausgabe erst C_POS erhöht, und dann das Zeichen ausgibt. Wärend in Mode 2 nur ein Byte addiert werden muß, werden im Mode 1 wegen der doppelten Zeichenbreite 2 Bytes addiert.

* Die Variablen MAX_CRX und MAX_CRY geben an, wie viele Zeichen (MAX_CRX) eine Zeile hat, und wie viele Zeilen (MAX_CRY) das Bild hat. Beide Variablen enthalten stets die für Mode 2 gültigen Werte, ganz gleich welcher Bildschirm-Modus gerade aktiv ist. Beispiel: Befindet sich der CPC im Mode 1, und hat 40 Zeichen pro Zeile und 25 Zeilen, dann hat die Variable MAX_CRX trozdem den Wert 80. Genaugenommen enthält MAX_CRX die Anzahl der Bytes pro horizontaler Zeile.

Beide Variablen werden von einigen Steuerzeichen benutzt. Ihr Inhalt sollte also der Realität entsprechen. Übrigends werden beide Variablen automatisch von den OS Funktionen S80X25, S68X30 und S64X32 angepasst.

* Variable RAMCHAR gibt an, ob ein RAM oder ein ROM Zeichensatz aktiv ist, also ob das Lower ROM aus- oder eingeschaltet werden soll.

Enthält RAMCHAR Bit 3 den Wert 0, dann wird der Zeichensatz aus dem ROM verwendet (unteres ROM an).

Enthält Bit 2 von RAMCHAR den Wert 1, so ist ein RAM Zeichensatz aktiv (unteres ROM aus). In diesem Fall muss ein Zeichensatz zwischen &3800 und &3FFF vorhanden sein.

Die Bits 0 und 1 von RAMCHAR enthalten den Bildschirm-Modus 0-2(, 3).

Beispiel Code (am Anfang jedes Programmes sinnvoll):

```
LD  BC,&7F81     ;MODE 1 einschalten und ...
                 ;(Wert &7F82 für Mode 2)
LD  A,(RAMCHAR) ;RAM oder ROM Zeichensatz ...
AND A,&04        ;Bit 2 isolieren.
                 ;Gesetzt => unteres ROM einblenden.
OR  A,C          ;Bit 2 und &81 (bzw. &82) mischen
OUT (C),A        ;unteres RAM oder ROM einblenden
```

Die Ausgabe einzelner Zeichen

* Mode 1: Im Bildschirm-Modus 1 lassen sich einzelne Zeichen in allen drei Farben/Pens ausgeben. Außerdem existiert noch eine Routine die Zeichen halb mit Ink 1 und halb mit Ink 2 ausgibt, dieses Verfahren stellt die schnellste Mode 1 Zeichenausgabe dar.

PRI0GG ==> Ein Zeichen wird mit Pen 1 (normal Gelb) dargestellt.
PRI0BB ==> Ein Zeichen wird mit Pen 2 (normal Blau) dargestellt.
PRI0RR ==> Ein Zeichen wird mit Pen 3 (normal Rot) dargestellt.
PRI0GB ==> Ein Zeichen wird mit Pen 1 und 2 (Gelb & Blau) dargestellt.

* Mode 2: In Mode 2 lassen sich einzelne Zeichen mit verschiedenen Attributen ausgeben:

PR_2 ==> gibt ein Zeichen ganz normal aus.
PR_2I ==> gibt ein Zeichen invertiert aus.
PR_2K ==> gibt ein Zeichen kursiv aus.
PR_2U ==> gibt ein Zeichen unterstrichen aus.
PR_2D ==> gibt ein Zeichen durchgestrichen aus.

Steuerzeichen werden dabei von keiner dieser (in ROM A enthaltenen) OS Funktionen beachtet, weder in Mode 1 noch in Mode 2.

Die Ausgabe von Strings definierter Länge

Strings sind Zeichenketten definierter Länge. Steuerzeichen werden wie ganz normale Zeichen dargestellt. Die Stringausgabe befindet sich in ROM A des FutureOS.

* Mode 1: Strings können mit allen drei Pens und in der Pen 1 + Pen 2 Mischfarbe ausgegeben werden:

STR_GG ==> Ink 1 (normal gelb)
STR_BB ==> Ink 2 (normal blau)
STR_RR ==> Ink 3 (normal rot)
STR_GB ==> Ink 1 + 2 (normal gelb, blau)

* Mode 2: Strings können mit allen fünf Attributen ausgegeben werden:

STR_2 ==> normal
STR_2I ==> invertiert
STR_2K ==> kursiv
STR_2U ==> unterstrichen
STR_2D ==> durchgestrichen

Die Ausgabe von Terms/Strings variabler Länge

Ein Term ist ein String variabler Länge, eine Zeichenkette, die durch das Byte &00 oder &1A terminiert wird. Alle 32 Steuerzeichen werden auch als solche behandelt.

Wie immer muß ein Zeichensatz ab Adresse &3800 vorhanden sein. Es ist auch möglich den ROM Zeichensatz zu nutzen (L-ROM einblenden).

* Mode 1: Auch bei den Terms kann man zwischen den drei Pens und der Mischfarbe aus Pen 1 + 2 wählen:

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

Mode 2: Die fünf Attribute stehen auch bei den Mode 2 Terms zur Verfügung:

TERM_2 ==> normal
TERM_2I ==> invertiert
TERM_2K ==> kursiv
TERM_2U ==> unterstrichen
TERM_2D ==> durchgestrichen

Mit PR2GR können 8-fach gezoomte Riesen-buchstaben ausgegeben werden.

Die Diskettenverwaltung

Die Massenspeicher-Verwaltung des FutureOS wurde mit hohem Aufwand programmiert. Die FDC- und HDC-Verwaltung des FutureOS ist zum AmsDOS und den üblichen Disketten- und Festplatten-Formaten kompatibel. Es werden Standartformate wie Data, System, IBM und Vortex unterstützt. Trotz der vollständigen Kompatibilität ist die Diskettenverwaltung unter FutureOS gänzlich anders organisiert, als die des AmsDOS oder CP/M. Die OS Funktionen der Massenspeicher-Verwaltung sind fast alle im ROM B und in Teilen des ROM C lokalisiert.

Gegenüber AmsDOS und CPM ist die Diskettenverwaltung des OS erweitert. Beispiele sind der Dateiheader von nicht-ASCII Dateien und die gleichzeitige Verwaltung von bis zu acht Floppy Disk Laufwerken. Dabei besteht die Möglichkeit am internen FDC vier LWs zu betrieben. Diese werden unter FutureOS als A, B, C und D bezeichnet. Zur Verwendung von C und D ist allerdings ein kleiner Hardware-Patch nötig. Als interner FDC wird derjenige FDC bezeichnet, der im CPC6128 eingebaut ist. Basisadresse des Hauptstatur-Register (HSR) ist &FB7E.

Am externen FDC können ebenfalls vier Laufwerke angeschlossen werden. Der externe FDC ist derjenige, der von der Firma Vortex mit den F1-D bzw. F1-S Laufwerken geliefert worden ist. Basisadresse HSR = &FBF6.

Die Architektur der FDC Verwaltung:

Die Architektur der FDC Verwaltung unterscheidet sich grundlegend von der des AmsDOS oder CP/M.

Unter FutureOS werden alle eingelesenen DIRs (Inhaltsverzeichnisse), im RAM gepuffert. Bevor man mit Dateien arbeiten kann muß ein DIR eingelesen werden. Die RAM Pufferung von DIRs beschleunigt die Disketten-Operationen. Läd oder speichert man von oder auf Diskette, dann braucht der Schreib/Lesekopf nur die entsprechenden Daten-Spuren anfahren, und nicht zusätzlich das DIR. Zusammen mit den schnellen Spurlade-Routinen macht das das Lesen und Schreiben um ein vielfaches schneller als AmsDOS. Zusätzlich wird bei Massenoperationen (z.B. Löschen, Umbenennen usw.) das DIR nur einmal, am Ende der Operation, auf Diskette geschrieben.

AmsDOS läd Dateien Block für Block. FutureOS erstellt zuerst eine Track & Sektor Tabelle aus den Blocknummern einer Datei. Anschließend wird die Datei am Stück geladen. Die Berechnung der Tabelle erfolgt wärend des Hochlaufens des Motors des Laufwerks bzw. wärend des ersten Spurwechsels, sie kostet also real keine Zeit. Unter FutureOS existiert keine Hochlaufzeit für Disk-Laufwerke, stattdessen werden die Motoren gestartet, und sobald das LW bereit meldet beginnt der Disk-Zugriff.

Jedes Laufwerk hat seine eigene SWZ=Spurwechselzeit (Step-rate-time). Schließt man mal ein schnelles Laufwerk an, so kann die SWZ im RAM oder ROM angleichen werden. Die SWZ

der Laufwerke A bis H steht ab DSWZ = &BA50 im RAM und ab RSWZ = &C017 im ROM B. Es folgt die Aufschlüsselung der SWZ-Code Bytes:

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

Normale 3 Zoll LWs laufen mit 12 ms (selten 10 ms). 3.5 Zoll und 5.25 Zoll Laufwerke (80 Spuren) laufen mit 4 ms. Je geringer die SWZ eines Laufwerks ist, desto schneller ist es.

Die OS Funktionen der FDC Verwaltung:

FutureOS stellt unterschiedliche Funktionen zur Disketten-Verwaltung zur verfügung.

Die Low-Level Funktionen erlauben direkten Zugriff auf den FDC. Da sind Funktionen zum lesen und schreiben von Spuren oder Verzeichnissen etc. Die High-Level Funktionen gestatten das lesen oder schreiben von Dateien, das Formatieren oder einen Datei-Heade anzuzeigen etc.

Die OS Funktionen der FDC- und der Massenspeicher-Verwaltung sind detailiert in den Dateien „API-B-DE.TXT.docx“ und „API-C-DE.TXT.docx“ beschrieben.

Für den normalen Gebrauch sind die OS Funktionen LADEN, LADE_N, SICHRE des C ROMs von Interesse.

Aufbau eines 128 Bytes Datei-Kopfes unter Amsdos und FutureOS

AMSDOS:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00	UU	N0	N1	N2	N3	N4	N5	N6	N7	E0	E1	E2	00	00	00	00	- User, Name,Extension
10	00	00	TT	00	00	SL	SH	00	LL	LH	AL	AH	00	00	00	00	- Typ,Start,Länge,ASta
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	- unbenutzt
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	- unbenutzt
40	LL	LH	00	PL	PH	??	??	??	??	??	??	??	??	??	??	??	- Länge, Prüfsumme
50	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	- unbestimmt
60	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	- unbestimmt
70	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	??	- unbestimmt

Dabei bedeutet Typ = Datei-Typ, St = Start-/Lade-Adresse ,Len = Datei-Länge und A.St = Auto-Start-Adresse.

Unter FutureOS ist der Datei-Kopf erweitert. Die von AMSDOS genutzten Bytes behalten ihre Funktion. Die bisher unbenutzten Bytes werden ebenfalls benutzt.

FutureOS:

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

FutureOS - 128 Byte Datei-Header Byte für Byte:

```
Byte(s) ! Bedeutung
--------!-----------------------------
     00 ! Usernummer der Datei &00..&FE
01 - 08 ! Dateiname (8 Bytes)
09 - 0B ! Extension (3 Bytes)
0C - 11 ! I00 bis I05 (6 Bytes) Icondaten
     12 ! Dateityp: Basic &00, Binär &02, Prowort &0A und bei
        ! Protect je + 1 z.B.: geschütztes Basic &01
        ! Unter FutureOS zusätzlich + &80 (8. Bit gesetzt)
     13 ! Horizontale Ausdehnung, X-Richtung falls Semi-Grafikicon.
     14 ! Vertikale Ausdehnung, Y-Richtung falls Semi-Grafikicon.
15 - 16 ! Low, Highbyte Start,Ladeadresse der Datei (16 Bit, 64KB).
     17 ! Nummer des 16K Blocks der Start,Ladeadr. &C0,&C4,&C5,..,&FF.
18 - 19 ! Low, Highbyte der Dateilänge.
1A - 1B ! Low, Highbyte der Autostartadresse der Datei, sonst &0000
     1C ! Overbyte der Dateilänge. ==> Datei kann bis 16MB lang sein.
     1D ! Nummer des 16K Blocks der Autostartadr. &C0,&C4,&C5,..,&FF.
1E - 3F ! I06 bis I27 (34 Bytes) Icondaten
40 - 41 ! Low,High Len.d.Datei. WIEDERHOLUNG unt.16 Bit. Ignorieren!
     42 ! Icontyp Text oder Grafikicon.
43 - 44 ! Low, Highbyte der 16 Bit Prüfsumme (über Header-Bytes 0..66)
45 - 7F ! I28 bis I62 (59 Bytes) Icondaten
--------!---------------------------------
```

Es stehen &63 = 99 Bytes für Icon-Daten zur Verfügung. Das Icon-Typ-Byte (&42) bestimmt ob es sich um ein GrafikIcon (4 * 3 oder 6 * 2), ein Semigrafik-Icon oder um ein TextIcon handelt. Es existieren vier unterschiedliche Icontypen. Alle Icons werden in Mode 1 dargestellt, und zwar bei 32 Zeichen pro Zeile und 32 Zeilen.

Handelt es sich um einen Text (Icontyp &00), so kann dieser z.B. Aufschluss über den Dateiinhalt oder deren Verwendung geben. Ein Semigrafikicon (Icontyp &02) wird als ASCII-Rechteck ausgegeben, die Ausdehnung in X u. Y Richtung wird durch die entsprechenden Bytes &13 (X) und &14 (Y) bestimmt. Multipliziert man X und Y, so darf der Wert &63 = 99 nicht überschritten werden. Im Semigrafikicon können alle 256 verschiedenen Zeichen verwendet werden. Bei Text- und Semigrafik-Icons wird jedes Icon-Datenbyte als Zeichen interpretiert. Der einzige Unterschied besteht darin, dass ein TextIcon 'am Stück' ausgegeben wird, wogegen ein Semi-GrafikIcon als Rechteck ausgegeben wird.

Im Gegensatz dazu enthalten die echten GrafikIcons nur Grafikdaten. Grafikicons enthalten Mode 1 Daten. Entweder im Format 2 * 3 (Icontyp &01) oder im Format 3 * 2 Zeichen (Icontyp &03). Von den 99 Bytes werden also nur 4 * 3 * 8 = 96 Bytes verwendet. Die drei verbleibenden Bytes stehen zur freien Verfügung.

Applikationen unter FutureOS:

Informationen über die verschiedenen Arten von Programmen sind in dem Abschnitt 'Programm - Architektur' zu finden.

Um eine Applikation zu starten markiert man sie im DIR. Anschließend betätigt man das RUN - Icon (Hotkey X). Nun wird das erste, markierte Programm geladen und gestartet. Das Programm muß einen Header besitzen, dieser gibt Lade- und Start- Adresse an.

Um ein Programm größer als 44 KB zu erstellen, sollte man es als E-RAM Programm konzipieren, und durch einen kleinen Lader starten, der das eigentliche Programm in die freien E-RAM-Blöcke läd.

Die 16 KB RAM Blöcke des Erweiterungs-RAMs werden unter FutureOS zwischen &4000 und &7FFF eingeblendet. Ein E-RAM Programm befindet sich ausschließlich im Erweiterungs-RAM. Es kann - in Abhängigkeit vom angeschlossenen E-RAM - bis zu 4 MB groß sein.

OS Funktionen zur Verwaltung von E-RAM

FutureOS bietet eine ausführliche Speicherverwaltung von bis zu 4 MB Erweiterungs-RAM (E-RAM). Die dazu genutzten RAM Variablen des OS wurden bereits in Abschnitt III (Seite 18) besprochen. Weiterhin stellt FutureOS eine Reihe von OS Funktionen zur Verfügung, die es einer Applikation leicht machen mit E-RAM zu arbeiten.

Die kleinste vom OS unterstützte Einheit sind dabei 16 KB Blöcke. Diese Größe wurde deshalb gewählt, weil sie der kleinsten Größe von der Hardware unabhängig schaltbarer RAM-Bereiche entspricht. 16 KB erscheinen viel, wenn man nur 64 KB E-RAM hat, jedoch sind es bei 4 MB bereits 256 Blöcke von 16 KB. Heute, im 21. Jahrhundert haben die meisten CPCs 512 KB E-RAM (ein X-MEM kostet nur 30 Euro!), und somit sind 32 Blöcke vorhanden. In diesem Licht erscheinen 16 KB als guter Kompromiss, der sich außerdem an den Gegebenheiten der Hardware orientiert.

Es gibt drei Arten von OS Funktionen zur E-RAM Verwaltung:

1) OS Funktionen die zeigen, wie viel E-RAM vorhanden / nutzbar ist
- FESB : Berechnet freien Erweiterungs-Speicher (erste 512 KB E-RAM)
- TXR4M : Testet wie viel E-RAM angeschlossen ist (bis 4 MB)
- GTPRB : Erstellt eine Tabelle freier E-RAM Blöcke

2) OS Funktionen die E-RAM für Applikationen belegen bzw. freigeben
- EFER : Weist einem Programm einen 16 KB E-RAM Block zu
- FER7F : Gibt 16 KB E-RAM Block in XRAM_?? Variablen frei
- SSB0 : Sichert das untere 16 KB RAM (&0000-&3FFF) im E-RAM
- RRB0 : Restauriert das untere 16 KB RAM (&0000-&3FFF) aus E-RAM

3) OS Funktionen für Berechnungen im Bezug auf E-RAM
- NXX_ERM : Selektiere nächstes 16 KB E-RAM (aus 4 MB) nach (AKT_RAM)
- NXT_ERM : Selektiere nächstes 16 KB E-RAM (aus 4 MB) nach BC

- LXX_ERM : Selektiere 16 KB E-RAM (aus 4 MB) vor E-RAM in (AKT_RAM)
- LST_ERM : Selektiere 16 KB E-RAM (aus 4 MB) vor E-RAM in BC

- E2XRAM : Rechnet physikalisches E-RAM (&C4 - &FF) in Zeiger auf XRAM_?? Variable um (erste 512 KB)

- XR2ER : Rechnet Zeiger auf E-RAM Variable (XRAM_C4, C5, ..FF) in physikalischen E-RAM Block (&C4, &C5..&FF) um

- BJKG : Rechnet Zeiger auf E-RAM Variable (XRAM_C4, C5, ..FF) in physikalischen E-RAM Block (&C4, &C5..&FF) um. Setzt B = &7F

Ausgabe von Zeichen und Strings auf dem Drucker

Ein Betriebssystem ist nur dann etwas wert wenn man die Ergebnisse seiner Arbeit auch zu Papier bringen kann - so oder so ähnlich hörte es sich im letzten Millenium an. Nach heutiger Sichtweise ein überkommener Standpunkt. Oder doch nicht?

In jedem Fall bietet FutureOS alle was zur Ausgabe von Zeichen, Strings oder Grafik auf einen Drucker notwendig ist. Text lässt sich bequem mit 7 Bit ausgeben, deshalb hat der CPC6128 auch nur einen 7 Bit Druckeranschluss. Grafiken hingegen lassen sich effizienter mit 8 Bit ausgeben. Deshalb hat der 6128 Plus einen 8 Bit Druckeranschluss. Doch auch für den CPC6128 kann das 8. Druckerbit bequem nachgerüstet werden: Dazu trennt man D7 des Druckerports von Masse und verbindet es mit Pin 12 (WR Data) der 8255 PIO des CPC. Das 8. Bit wird also genau so gesetzt bzw. gelöscht wie man Daten auf die Kassette schreibt. Doch der Anwednder bzw. die Applikation muss sich darum nicht kümmern, das machen die OS Funktionen des FutureOS:

Allgemeine OS Funktionen zum Thema Drucker:
- XW_DR : Wartet bis der Drucker bereit ist ein Zeichen zu empfangen

OS Funktionen für den Ausdruck im 7 Bit Modus:
- DRZ7 : Druckt ein 7 Bit Zeichen
- DRS7 : Druckt einen String (7 Bit)

OS Funktionen für den Ausdruck im 8 Bit Modus:
- DRZP8 : Druckt ein 8 Bit Zeichen - Nur CPC Plus!
- DRZO8 : Druckt ein 8 Bit Zeichen - Nur CPC 6128 mit Hardware-Patch!

- DRSP8 : Druckt einen String (8 Bit) - Nur CPC Plus!
- DRSO8 : Druckt einen String (8 Bit) - Nur CPC 6128 mit Hardware-Patch!

Die Konfiguration des FutureOS gibt darüber auskunft, ob im 8 Bit Modus gearbeitet werden kann. Das 7. Bit (also Bit 6) der RAM Variable KF_SIO ist gelöscht, wenn nur 7 Bit zur Vefügung stehen. Oder gesetzt, wenn im 8 Bit Modus gearbeitet werden kann.

Nützliche OS Funktions-Aufrufe

In diesem Kapitel sind einige interessante OS Funktions-Aufrufe kurz beschrieben. Weitere Informationen sind in den ‚API-?-DE.DOK' Datein der API-Dokumentation zu finden.

Laden von Dateien

- LADE_N : Läd Datei, die nur durch Laufwerk, Usernummer, Name und Extension definiert ist (Diskette oder Festplatte)
- LADEN : Läd die erste markierte Datei von Diskette oder Festplatte
- TEILA/B : Teilweise Laden einer Datei

Speichern von Dateien

- SICHRE : Sichert Datei, definiert durch Laufwerk, User, Name und Extension (Diskette oder Festplatte)
- TEISI/SK : Teilweise Sichern einer Datei

Arbeit mit Datei-Headern

- SHED : Zeigt einen FutureOS Datei-Header in Mode 1 (32*32)
- LADAH : Läd nur den Header einer Datei
- TST_HED : Berechnet die Prüfsumme eines Datei-Headers

Dateien und Bilder anzeigen

- TYSAZ : Stellt eine Seite Text im Mode 2 auf dem Bildschirm dar
- ZEIBI/J : Zeigen eines Bildes auf dem Bildschirm. Normale 17 KB Bildschirme bzw. komprimierte OCP Bildschirme MODE und Bildschirm-Format durch Cursortasten einstellbar
- OCPC0 : OCP .SCR Datei entkomprimieren und auf Bildschirm anzeigen

Bildschirmformate einstellen

- S64X32 : Setzt den Bildschirm auf 64 Spalten und 32 Zeilen
- S68X30 : Setzt den Bildschirm auf 68 Spalten und 30 Zeilen
- S80X25 : Setzt den Bildschirm auf 80 Spalten und 25 Zeilen

Verwaltung von Echtzeituhren

Lesen:

- R_RTC : Liest Zeit und Datum von angeschlossener Echtzeituhr ein (Hauptfunktion)
- R_SFRT : Lese Zeit und Datum aus SF-II Echtzeituhr
- R_SF3RTC : Lese Zeit und Datum aus SF-III Echtzeituhr
- R_LS3RTC : Lese Zeit und Datum aus LambdaSpeak III / FS Echtzeituhr
- R_NRTC : Lese Zeit und Datum aus der Nova Karte

Schreiben:

- S_RTC : Datum und Zeit werden in alle RTCs geschrieben (Hauptfunktion)
- S_SFRT : Schreibe Zeit und Datum in SF-II Echtzeituhr
- S_SF3RTC : Beschreibt die RTC des SYMBiFACE III mit Zeit und Datum
- S_LS3RTC : Beschreibt die RTC des LambdaSpeak III bzw. FS mit Zeit und Datum
- S_NRTC : Beschreibt die RTC der Nova Karte mit Zeit und Datum

Mathematik

- MUL88 : Schnelle 8 * 8 Bit Multiplikation
- DIV88 : Schnelle 8 / 8 Bit Division

Verwaltung von Mäusen und weiteren HID Geräten

- R_ALM : Liest die Positionsdaten einer angeschlossenen proportionalen Maus ein
- R_PS2 : Liest die Positionsdaten der PS/2 Maus des SYMBiFACE II
- R_USB3 : Liest die Positionsdaten der USB Maus des SYMBiFACE III
- R_ALB : Liest die Positionsdaten der USB Maus des Albireo
- R_MPM : Liest die Positionsdaten der MultiPlay Mäuse
  (siehe auch OS Funktionen R_MPM1 und R_MPM2)
- G_GP2 : Abfrage des Hegetron Grafpad II

Umwandlung von Bytes und Daten, BCD und Binär

- CC2N : Konvertiert zwei ASCII Zeichen in einen 8 Bit Wert
- N_2_2C : Konvertiert ein Byte in zwei ASCII Zeichen
- HAUT : Schreibt einen hexadezimalen 8 Bit Wert auf den Bildschirm
- BIN2BCD : Eine binäre Zahl (0-99) wird in eine BCD Zahl umgewandelt
- BCD2BIN : Eine BCD Zahl wird in eine binäre Zahl (0-99) umgewandelt

Speicherbereiche kopieren, füllen oder löschen

- F_FILL8 und F_FILL6 : die schnellste Möglichkeit das RAM des CPC mit einem 8 oder 16 Bit Wert zu füllen. Pro Byte werden nur 2 µs benötigt
- F_MOVE : verschiebt Speicherbereiche sehr schnell. Pro Byte werden nur 5 µs Zeit benötigt
- LESC : Schnellstes Bildschirmlöschen &C000 bis &FFFF, wie CLS

Debugging

- F_DUMP : Stelle einen Speicherauszug/DUMP auf dem Bildschirm dar
- EDIT : Ein Speicherbereich von &01E0 Bytes (aktuell angezeigt) wird im Hexadezimal-Modus editiert
- F_PORT : Anzeigen und editieren aller Ports des CPC
- CARET : Dies ist der Einsprung in den Maschinen - Monitor, alle Z80 Register werden in ihre RAM-Variablen gesichert

Klangverwaltung

- MAUS : Schalte jegliche Klang-Ausgabe des PSG aus
- S_PSG : Schreibt Daten in den PSG

Hardware auf Vorhandensein abfragen

- T_SF : Prüfe ob ein Symbiface II angeschlossen ist
- T_SF3 : Funktion tested ob das SYMBiFACE III vorhanden ist
- W_SF3 : Warten bis das SYMBiFACE III Bereitschaft meldet

Rücksprung ins FutureOS

- TUR_E : Ist DER Standart-Einsprung oder Rücksprung ins OS
- TUR_D : entspricht TUR_E, setzt aber zusätzlich alle DIRs zurück. Alle DIRs werden verworfen und TURBO_A bis M initialisiert. Das HinterGrundBild wird zurückgesetzt
- KLICK : Rücksprung ins OS, die Icons (obere Bild-Hälfte) dürfen nicht manipuliert sein. Sinnvoll bei kleinen Programmen, die lediglich die untere Hälfte des Desktops ändern

Weitere OS Funktionen

- FMD32 : Sucht die erste markierte Datei aller eingelesener DIRs

- GET_DIR : Liest Inhaltsverzeichnisse aller markierten LW's ein

- LTERM_2 : Gibt einen Text auf dem Bildschirm und via LambdaSpeak aus

VII. Anhänge

Icons

Das Desktop macht ausgiebig Gebrauch von diversen Icons. Alle Icons und Icon-Sätze zusammen verbrauchen ca. 9,5 KB im D-ROM. Nun einige Informationen zu den normalen Icons. Wie erwähnt befinden sich ihre Grafikdaten im D-ROM. Ihre Adressen lassen sich in der Datei ‚#EQU-API.DEU' nachschlagen.

Jedes Icon ist für die Verwendung in Mode 2 gezeichnet worden. Es ist 6 Mode-2 Zeichen breit und 3 Zeichen hoch. Es muß jedes Pixel am äußeren Rand gesetzt sein. Wollen Sie eigene Icons eingebaut haben so ist dies einfach möglich, verändern Sie dazu einfach denentsprechenden ROM-Bereich, aber Vorsicht..

Das Icon ist folgendermaßen im ROM abgelegt: Zuerst kommt das Byte von ganz links oben, dann das Byte rechts daneben - bis alle sechs Bytes dieser Zeile durch sind. Darauf folgen die sechs Bytes der nächstunteren Rasterzeile - usw. bis alle Rasterzeilen von oben nach untern durchlaufen sind. Aber Achtung!! Die oberste und unterste Raster-Zeile fehlt. Im ROM sind also nur 22 Rasterzeilen abgelegt. Denn die oberste und unterste Zeile wird immer als Strich dargestellt - je sechs Bytes &FF. Dies macht die Routine ICON6ON des D-ROMs.

Falls es ihnen zu komplex ist Icons zu manipulieren, dann sagen Sie mir bescheid, ich kann ihnen dann ihr eigenes Icon schon einbauen.

Iconsätze

Die Icons der Icon-Zahlen-Sätze unterscheiden sich durch ihre Breite von den normalen Funktions-Icons. Sie sind nur drei Bytes breit. Im ROM sind sie analog zu den normalen Icons abgelegt: Zuerst kommt das Byte von ganz links oben, dann das Byte rechts daneben und dann das dritte rechts, oben. Darauf folgen die drei Bytes der nächst-unteren Rasterzeile - usw. Die oberste und unterste Raster-Zeile fehlt. Im ROM sind also nur 22 Rasterzeilen abgelegt. Denn die oberste und unterste Zeile wird immer als Strich dargestellt. Dafür ist die Routine ICON3ON des D-ROMs zuständig.

Es existieren zwei Icon-Zeichensätze, die jehweils die Ziffern von Null bis Neun enthalten.

Benutzt man ICON3ON, dann ist es problemlos möglich sich dieser großen Zahlen zu bedienen.

Der Mauspfeil

Wärend die CPCs der 'old Generation' ein Software-Sprite als Mauspfeil benutzen, übernimmt diese Funktion bei den Plus-CPCs das Hardware-Sprite 0 ab Adresse &4000 im MM/ASIC Bereich.
Wünscht man sich einen neuen Mauspfeil, dann kann man ihn ins D-ROM einbauen.
Bei der FutureOS Version für CPCplus ist das Hard-Sprite von 256 auf 128 Bytes komprimiert.
Bei dem CPCs old Generation ist das Soft-Sprite unkomprimiert.

Die möglichen Eingabemedien

Um mit FutureOS zu kommunizieren, können diverse Medien verwendet werden. Die Multiscan Umgebung erkennt welches Medium benutzt wird.

- Joystick 0 oder Joystick 1
- Cursortasten + Copy + kleines Enter
- Joystick-kompatible Mäuse oder Atari-ST Maus
- Marconi Trackball, Joy-kompatible Trackbälle oder Atari-Trackball
- Analog-Joystick (nur 6128plus)
- LightPen + Copy (bisher nur 6128plus)

Unterschiede vom CPC6128 zum 6128 Plus

Vom FutureOS existieren zwei leicht verschiedene Versionen, eine für die CPC6128 der 'old Generation', die andere für die neuen CPCs 464/6128 plus. Als Programmierer des OS habe ich die Unterschiede denkbar gering gehalten.

Ein FutureOS Programm ist in jedem Fall unter beiden OS-Versionen lauffähig, da sich die Unterschiede nicht in der Programm-Umgebung abzeichenen.

Die Version für die alten CPCs läuft natürlich auch auf den CPCs der Plus Serie.

Der Hauptunterschied beider Versionen liegt wohl noch darin, daß bei den neuen CPCs ein Hardware-Sprite (15 Farben oder durchsichtig) als Mauspfeil benutzt wird, wogegen bei den alten CPCs ein Software-Sprite (monochrom) dessen Funktion übernimmt.

Nur bei den CPCs der 'old Generation' können Atari-ST Maus bzw. Atari-ST Trackball verwendet werden. Der Analog-Joystick (und der LightPen) dagegen kann nur bei den Plus-CPCs benutzt werden.

Abgesehen davon kann bei den neuen CPCs im Maschinen-Monitor außer der RAM- und ROM- Konfiguration auch der Status der Memory-Mapped-Baugruppen eingestellet werden (MM einblenden Ja/Nein).

Benutzt man das FutureOS auf einem CPC464, dann funktioniert die Soft-Sprite-Verwaltung nicht. Dies hängt wohl damit zusammen, daß eine bestimmte RAM-Konfiguration (&C3) nicht eingestellt werden kann. Das Soft-Sprite hinterläßt Schlieren auf dem Bildschirm. Abhilfe schafft eine RAM Erweiterung die die RAM Konfiguration &7FC3 unterstützt.

Definition der CPC Klassen

Klasse 0 CPC: 64 KB RAM, kein Laufwerk (CPC-464)

Klasse 1 CPC: 64 KB RAM, 3" Laufwerk (CPC-664)

Klasse 2 CPC: 128 KB RAM, 3" Laufwerk (CPC6128)

Klasse 3 CPC: 128 KB RAM, 3" Laufwerk, 2. Laufwerk: 80 Spuren, DS

Klasse 4 CPC: 320 KB RAM, 3" Laufwerk, 2. Laufwerk: 80 Spuren, DS

Klasse 5 CPC: 576 KB RAM, 3" Laufwerk, 2. Laufwerk: 80 Spuren, DS

Klasse 6 CPC: 576 KB RAM, 3" & 2. Laufwerk (80/DS), 20 MB Festplatte

Klasse 7 CPC: 2048 KB RAM, 4 interne & 4 externe Laufwerke, 20 MB HD

Klasse 8 CPC: 4096 KB RAM, 2-8 Laufwerke, Festplatte mit min. 20 MB

Die Menge des vorhandenen Speicherplatzes (lesen/schreiben) definiert die Klasse des CPC. Angeschlossenes EPROM (RO!) hat keinen Einfluss.

FutureOS benötigt mindestens einen Klasse 2 CPC. Es läuft am besten auf einem Klasse 4 CPC oder höher.

Die Hardware des CPC

Die Portadressen der Amstrad CPC Computer:

Im folgenden habe ich alle Portadressen der CPCs aufgelistet, die mir momentan bekannt sind. Diese Liste erhebt keinen Anspruch auf Vollständigkeit.

Ein XX bedeutet, daß das entsprechende Byte einen beliebigen Inhalt annehmen kann.

```
- &7FXX : Gate Array                                  schreiben / -

- &BCXX : 6845 CRTC Adress-Register                   schreiben / -
- &BDXX : 6845 CRTC Daten-Register                    schreiben / -
- &BEXX : 6845 CRTC Status-Register                           - / lesen
- &BFXX : 6845 CRTC Video-Adress-Register                     - / lesen

- &DFXX : ROM selektieren                             schreiben / -

- &EFXX : Drucker Port                                schreiben / -

- &F4XX : 8255 PIO Port A                             schreiben / lesen
- &F5XX : 8255 PIO Port B                                     - / lesen
- &F6XX : 8255 PIO Port C                             schreiben / -
- &F7XX : 8255 PIO Steuer-Register                    schreiben / -

- &F8B0 : Vidi-CPC Video-Digitizer, CRTC Index        schreiben / ?
- &F8B1 : Vidi-CPC Video-Digitizer, CRTC Daten        schreiben / ?

- &F8E0 : Z80 STI Indirect Data Register              schreiben / lesen
- &F8E1 : Z80 STI Gen. Purpose I/O Interrupt          schreiben / lesen
- &F8E2 bis & F8E7 Interrupt, nicht benutzbar
- &F8E8 : Z80 STI Pointer Vector Register             schreiben / lesen
- &F8E9 bis & F8EB Timer, nicht benutzbar
- &F8EC : Z80 STI USART Control Register              schreiben / lesen
- &F8ED : Z80 STI Receiver Status Register            schreiben / lesen
- &F8EE : Z80 STI Transmitter Status Register         schreiben / lesen
- &F8EF : Z80 STI USART Data Register                 schreiben / lesen

- &F8E2 bis &F8E4: Dobbertin Eprommer 4003                    ? / ?
- &F8F2 : Dobbertin Eprommer 4003                             ? / ?

- &F9B0 : Vidi-CPC, Konfig.(w), Capture(r)            schreiben / lesen

- &F9FC bis &F9FE: Otten & Fecht 1 MB RAM-Disc                ? / ?

- &FA7E : Floppy Motor Steuerung                      schreiben / -

- &FADC : Z80-SIO / DART Kanal A Daten Reg.           schreiben / lesen
- &FADD : Z80-SIO / DART Kanal A Kontroll Reg.        schreiben / lesen
- &FADE : Z80-SIO / DART Kanal B Daten Reg. S         chreiben / lesen
- &FADF : Z80-SIO / DART Kanal B Kontroll Reg.        schreiben / lesen

- &FB7E : 765 FDC (intern) Status Register                    - / lesen
- &FB7F : 765 FDC (intern) Daten Register             schreiben / lesen

- &FBDC : 8253 Timer Zähler 0                         schreiben / lesen
- &FBDD : 8253 Timer Zähler 1                         schreiben / lesen
- &FBDE : 8253 Timer Zähler 2                         schreiben / lesen
- &FBDF : 8253 Timer Modus Wahl                       schreiben / -
```

- &FBE0 : Hard Disk Daten Port schreiben / lesen
- &FBE1 : Hard Disk Status, Reset schreiben / lesen
- &FBE2 : Hard Disk Select, Configuration schreiben / lesen
- &FBE3 : Hard Disk DMA, Interrupt schreiben / lesen
- &FBE4 : Hard Disk Reset schreiben / lesen

- &FBE0 bis &FBE3 und &FBE8 : dk'tronics RTC. ? / ?

- &FBEE : SSA1 dk'tronics Sprachausgabe-Modul schreiben / lesen

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

- &FBF6 : 765 FDC (Vortex,ext) Status Register - / lesen
- &FBF7 : 765 FDC (Vortex,ext) Daten Register schreiben / lesen

- &FD00 bis &FD21 : CPC-IDE / SYMBiFACE II schreiben / lesen

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

- &FF00 bis &FF2A : CPC-Booster(+) schreiben / lesen

Das Gate-Array des CPC

Das Gate Array des CPC ist für die RAM und die ROM Verwaltung zuständig, außerdem steuert es den Bildschirmmodus und ist für die Bildschirmfarben verantwortlich. Es ist NUR beschreibbar.

Die Basis-Adresse des Gate Arrays ist &7F??, es ist nur das High-Byte &7F entschiedent. Das Low-Byte ist irrelevant.

Das Gate Array hat vier interne 6-Bit Register. Es wird durch die beiden obersten Bits des Bytes selektiert, das an das Gate Array geschickt wird. Format des Bytes: %rrdddddd, rr=Register, d=Daten-Bit

Register 0 / %00dd dddd / &00,..,&3F ==> INK wählen
Register 1 / %01dd dddd / &40,..,&7F ==> dem INK Farbe zuweisen
Register 2 / %10dd dddd / &80,..,&BF ==> RAM/ROM Wahl, Mode-Wahl
Register 3 / %11dd dddd / &C0,..,&FF ==> RAM Banking

Register &00 (Null) des Gate Arrays:

Register &00 selektiert den INK, auf den sich das nächste Farb-Byte bezieht. Der Wert ist mit dem Basic-Befehl INK kompatibel, den Border selektiert man mit &10. Die Werte von &11 bis &3F sollte man nicht verwenden.

&00 ==> Für das nächste Farb-Byte wird der Untergrund selektiert.
&01 ==> Für das nächste Farb-Byte wird INK 1 selektiert.
&02 ==> Für das nächste Farb-Byte wird INK 2 selektiert.
... ==> ...
... ==> ...
&0F ==> Für das nächste Farb-Byte wird INK 15 selektiert.
&10 ==> Für das nächste Farb-Byte wird der Rand/Border selektiert.

Register &40 (Eins) des Gate Arrays:

Hat man zuvor mit Register Null des Gate Arrays den INK bzw. Rand selektiert, so kann man ihm über Register Eins einen Farbert zuweisen.

Aber Achtung, die Farb-Werte sind nicht mit den Basic-Farbwerten kompatibel. Folgende Tabelle zeigt welcher Gate-Array Code dem Basic-Wert entspricht.

Real-Zahlen-Werte für INK Farbe, &??
Farbe 0 - &54 Farbe 1 - &44 Farbe 2 - &55 Farbe 3 - &5C
Farbe 4 - &58 Farbe 5 - &5D Farbe 6 - &4C Farbe 7 - &45
Farbe 8 - &4D Farbe 9 - &56 Farbe 10 - &46 Farbe 11 - &57
Farbe 12 - &5E Farbe 13 - &40 Farbe 14 - &5F Farbe 15 - &4E
Farbe 16 - &47 Farbe 17 - &4F Farbe 18 - &52 Farbe 19 - &42
Farbe 20 - &53 Farbe 21 - &5A Farbe 22 - &59 Farbe 23 - &5B
Farbe 24 - &4A Farbe 25 - &43 Farbe 26 - &4B

Beispiel-Listings um den Rand/Border auf Schwarz und INK 1 auf Hellblau (2) einzustellen.

Basic:
```
10 INK 1,2
20 BORDER 1
```

Assembler:
```
LD BC,&7F10 ;BORDER selektieren
OUT (C),C
LD BC,&7F44 ;Farbe für BORDER einstellen
OUT (C),C

LD BC,&7F01 ;INK 1 selektieren
OUT (C),C
LD BC,&7F55 ;Farbe für INK 1 einstellen
OUT (C),C
```

Register &80 (Zwei) des Gate Arrays:

Bei diesem Register ist es nötig alle Bits aufzuschlüsseln, da fast jedes eine eigene Bedeutung hat.

Bits 7, 6 : %10XXXXXX ==> Diese Bits sind immer 1,0 und selektieren somit das Register Zwei des Gate Arrays.

Bit 5 : Reserviert, für die Plus CPCs, bitte Null senden.

Bit 4 : Wird eine Eins gesendet, dann wird der '52 Abtast Perioden Zähler' also der 'Interrupt-Zähler' gelöscht. Normalerweise sollte man eine Null senden.

Bit 3 : Wird eine Null gesendet, dann wird das aktuelle obere ROM zwischen &C000 und &FFFF eingeblendet. Sendet man eine Eins, so wird das RAM eingeblendet.

Bit 2 : Wird eine Null gesendet, dann wird das untere ROM zwischen &0000 und &3FFF eingeblendet. Sendet man eine Eins, so wird das RAM eingeblendet.

Bit 1, 0 : Diese beiden Bits dienen zur Auswahl des Bildschirm-Modus.
Dabei kommen folgende Bit-Kombinationen in Frage:
%10XX XX00 ==> Mode 0 aktivieren, 16 Farben
%10XX XX01 ==> Mode 1 aktivieren, 4 Farben
%10XX XX10 ==> Mode 2 aktivieren, 2 Farben
%10XX XX11 ==> Mode 3 aktivieren, 4 Farben, Modus ist illegal

Register &C0 (Drei) des Gate Arrays:

Dieses Register dient ausschließlich dem RAM - Banking, dabei existieren auch einige Spezial-Konfigurationen, die das interne 64K RAM umbelegt. Das E-RAM ist in acht Bänke mit je 64KB eingeteilt, die wiederum aus vier 16KB Blöcken bestehen.

%11???000 ==> blendet Standart 64KB ein.
%11BBB000 ==> 16K Block 4 einer Bank BBB ab &C000 einblenden.
%11BBB010 ==> Ganze 64K Bank BBB einblenden.
%11BBB011 ==> Block 4 der Bank BBB ab &C000 einblenden, der 16K Block des Standart-RAMs wird von &C000-&FFFF nach &4000-&7FFF geblendet.
%11BBB1NN ==> 16K Block NN einer Bank BBB wird zwischen &4000 und &7FFF eingeblendet.

Das Erweiterungs-RAM umfaßt maximal 512K, diese 512K lassen sich in 32 mal 16KB einteilen. Bei folgender Konfiguration wird jeder 16KB Block immer zwischen &4000 und &7FFF eingeblendet:

Konfiguration &C4 ==> blendet E-RAM Block 1 ein.
Konfiguration &C5 ==> blendet E-RAM Block 2 ein.
Konfiguration &C6 ==> blendet E-RAM Block 3 ein.
Konfiguration &C7 ==> blendet E-RAM Block 4 ein.

Konfiguration &CC ==> blendet E-RAM Block 5 ein.
Konfiguration &CD ==> blendet E-RAM Block 6 ein.
Konfiguration &CE ==> blendet E-RAM Block 7 ein.
Konfiguration &CF ==> blendet E-RAM Block 8 ein.

Konfiguration &D4 ==> blendet E-RAM Block 9 ein.
...
Konfiguration &D7 ==> blendet E-RAM Block 12 ein.
...
Konfiguration &FE ==> blendet E-RAM Block 31 ein.
Konfiguration &FF ==> blendet E-RAM Block 32 ein.

Die Blöcke 1 bis 8 sind im CPC6128 bereits eingebaut, das weitere RAM kann nachgerüstet werden.

Die PIO 8255 des CPC:

Die PIO hat im CPC vielfältige Aufgaben. Bevor wir ins Detail gehn, folgt eine kurze Beschreibung einiger ihrer Eigenschaften:

- die PIO stellt die einzige Möglichkeit dar um mit dem Soundchip in Verbindung zu treten.

- die gesammte Magnetbandverwaltung (Tape) wird von der PIO erledigt.

- mit Hilfe der PIO und des PSG (das ist der Soundchip) wird die Tastatur verwaltet.

- der Drucker wird auf Bereitschaft abgefragt.

- der Elektronenstrahlrücklauf kann abgefragt werden und noch vieles mehr...

Man sieht also, die PIO ist doch nicht mal so ohne und man kann damit auch einige Sauereien anstellen, wie z.B Daten über den Joystickport und die Tastatur auszugeben, dazu braucht man aber auch noch den Soundchip.

Die PIO ist ein Peripheriechip mit drei Kanälen, oder besser drei Ports mit je 8 Bit. Benennen wir diese 3 Ports mal mit A, B und C, um uns die Arbeit zu erleichtern. Ob über diese Ports nun Daten eingelesen werden, oder ob man Bytes hinausschickt wird im Steuerregister der PIO festgelegt. Für jeden Port läßt sich extra die Datenrichtung bestimmen. Bei Port C kann man für jedes einzelne Bit festlegen, ob Daten eingelesen oder ausgegeben werden sollen. Ins Steuerregister kann nur geschrieben werden. Die drei Ports können gelesen und beschrieben werden. Dies geschieht über folgende Adressen:

&F4XX - PIO Port A - lesen, schreiben
&F5XX - PIO Port B - lesen, schreiben
&F6XX - PIO Port C - lesen, schreiben
&F7XX - PIO Steuerregister - nur schreiben

Für XX kann hier ein beliebiger Wert stehen, da das Lowbyte bei der Adressierung nicht relevant ist.

Nun zuerst einige Informationen zu den einzelnen Ports und deren Bits, danach etwas zum Steuerregister.

Port A - Der gesammte Port A stellt eine Verbindung zum Datenbus des Soundchips dar. Will man Daten in ein Register des PSG schreiben, dann geschieht dies über diesen Port, genauso wie auch der Tastaturstatus über diesen Port eingelesen wird. Die Tastatur ist als Matrix, bestehend aus 10 Zeilen mit je 8 Bit angeordnet. Wie man nun eine dieser 10 Matrixzeilen wählt, oder das Zielregister im PSG selektiert wird weiter unten beschrieben. Port A wird also sowohl als Ein- als auch als Ausgabe Port verwendet.

Port B - In Port B hat fast jedes Bit eine andere Funktion. Dieser Port wird normalerweise als Eingabeport benutzt. Es folgt eine Aufschlüsselung nach den einzelnen Bits:

Bit 0 - VSYNC : Das Bit 0 ist normalerweise Null. In der Zeit in der ein Elektronenstrahlrücklauf stattfindet wird dieses Bit vom CRTC auf logisch Eins gesetzt. Tritt ein Interrupt auf, so kann man dieses Bit testen, und so z.B. das Scrolling mit dem Bildaufbau synchronisieren. Solch ein Elektronenstrahlrücklauf tritt im Normalfall (d.h. 50 Hz Monitorfrequenz) fünfzigmal pro Sekunde auf.

Bits 1, 2 und 3 - Lötkontakte 1..3 : Diese drei Bit sind auf der Platine des CPC als Jumper angeordnet. Für jedes dieser drei Bits ist eine eigene Drahtbrücke vorhanden, oder eben nicht vorhanden. Mit drei Bits kann man 8 verschiedene Zustände darstellen. Je nachdem welche dieser Bits nun gesetzt sind oder nicht, hat der CPC eine andere Einschaltmeldung.

Bit 4 - LK4 : Auch dieses Bit ist auf der Platine als Jumper angeordnet. Beim Reset prüft das Betriebssystem den Zustand dieses Bits und fährt das System entweder mit 50 Hz (Drahtbrücke gelötet) oder mit 60 Hz (keine Drahtbrücke) Monitorfrequenz hoch.

Bit 5 - EXPANSION : Dieses Bit spiegelt den Zustand des EXP Signals am Expansion Port. Man kann es verwenden um dem System Auskunft über eventuell vorhandene Erweiterungen zu geben.

Bit 6 - BUSY : Dieses Bit wird vom Drucker beeinflußt. Ist es Null, dann ist der Drucker bereit ein weiteres Byte zu lesen. Wenn dieses Bit allerdings auf Eins steht, dann können derzeit keine weiteren Bytes an der Drucker geschickt werden.

Bit 7 - READ DATA : Das achte Bit wird dazu verwendet um vom Kassettenrekorder Daten zu lesen. Da hier immer nur eine Null oder eine Eins gelesen werden kann, müssen die Daten auch in diesem dualen Format aufgezeichnet werden.

Port C : Auch in Port C ist die Bedeutung der einzelnen Bits sehr verschieden. Im Gegensatz zu Port B ist dieser Port ein reiner Ausgabeport.

Bits 3, 2, 1 und 0 : Die unteren 4 Bits werden in Zusammenhang mit der Tastaturverwaltung benötigt. über diese vier Bits wird die Tastaturreihe von 0 bis 9 selektiert, die über Port A eingelesen werden kann.

Bit 4 : Der Motor des Bandlaufwerks kann mit diesem Bit ein- oder ausgeschaltet werden. Will man von Kassette lesen, oder darauf schreiben, so setzt man dieses Bit auf 1 und schaltet damit den Motor des angeschlossenen Rekorders ein. Eine Null schaltet den Motor aus.

Bit 5 - WRITE DATA : Um auf Band zu schreiben verwendet man das sechste Bit. Es können auch hier nur zwei Zustände geschrieben werden, Null oder Eins. Dieses Bit dient auch als 8. Druckerbit, falls aktiv.

Bits 7 und 6 : Sie werden im Zusammenhang mit dem PSG benötigt. Da über Port A ja nur die Daten an den PSG geliefert oder von ihm gelesen werden, benötigt man weitere Bits um den Datenfluß zu kontrollieren.
Es existieren vier Zustände mit folgenden Bedeutungen:

```
Bits: 7 6 5 4 3 2 1 0
---------------------
----> 0 0 x x x x x x --> Warten / inaktiv
----> 0 1 x x x x x x --> Lesen
----> 1 0 x x x x x x --> Schreiben
----> 1 1 x x x x x x --> Registerwahl / Latch Adress
```

Setzt man also beide Bits (7,6) auf Null so wird der Datenbus des PSG hochohmig, dies hat zur Folge daß der PSG keine Daten mehr annimmt. Setzt man jedoch den PSG auf WARTEN, und schickt ihm dann ein Byte über den Port A und setzt ihn dann auf schreiben, dann akzeptiert er das Datenbyte. So wird der Datentransfer jedenfalls vom OS abgewickelt.

Ist Bit 7 auf Null, und Bit 6 auf Eins gesetzt, so ist der PSG auf LESEN geschaltet. Man kann nun über Port A das aktuelle Register des PSG auslesen.

Setzt man Bit 7 auf Eins, und Bit 6 auf Null, dann ist der PSG auf SCHREIBEN geschaltet, und man kann ihm, ins gerade aktive PSG-Register, Datenbytes übermitteln.

Sind beide Bits 6 und 7 auf Eins, dann werden sämtliche Daten die man über Port A an den PSG schickt als Registernummer interpretiert. Will man den PSG also auslesen oder beschreiben, dann wird auf diese Art zuvor die entsprechende Registernummer selektiert.

Wie setzt man aber nun die einzelnen Ports A, B und C auf Ein- oder Ausgabe? Dies geschieht über das Steuerregister. Es arbeitet ebenfalls bitorientiert. Das Steuerregister arbeitet in zwei Modis. Setzt man das oberste Bit auf Eins dann arbeitet das Steuerregister normal.

Setzt man aber das 8. Bit auf Null, dann sind spezielle Funktionen möglich.

Aber betrachten wir uns das Steuerregister zuerst in seiner Standartfunktion mit gesetztem 8. Bit. Es sei noch kurz erwähnt, daß die PIO die drei Ports intern in zwei Gruppen einteilt, und in 3 Modis arbeitet. Da beim CPC aber nur Modus 0 verwendet werden kann werde ich hier nicht näher darauf eingehen. Für Port A, Port B, und für die unteren bzw. die oberen 4 Bit von Port C existiert je ein Bit das die Datenrichtung des entsprechenden Ports angibt. Setzt man eines dieser Bits auf Null so bedeutet dies, daß der entsprechende Port als Ausgang geschaltet ist. Ist ein solches Bit auf Eins gesetzt, so kann man über den zugehörigen Port Daten einlesen. Nun aber der Aufbau des Steuerregisters:

Bits :	7	6	5	4	3	2	1	0
Bedeutung :	1	0	0	A	Ch	0	B	Cl
Standartwert :	1	0	0	?	0	0	1	0

Bit 4 gibt an, ob Port A auf Ein- oder Ausgabe geschaltet ist.

Bit 3 steht hier für die oberen vier Bit von Port C. Dieses Bit sollte stets auf Null stehen, um Daten ausgeben zu können.

Bit 1 setzt den Port B auf Ein- oder Ausgabe. Hier sollte stets eine Eins stehen, um Daten über diesen Port einlesen zu können.

Bit 0 bezieht sich auf die unteren 4 Bit von Port C, es sollte auch auf Ausgabe, also logisch Null stehen.

Soviel also zur normalen Funktion des Steuerregisters. Setzt man nun aber im Steuerregister das 8. Bit auf Null so hat es eine Spezialfunktion, diese bezieht sich exklusiv auf den Port C der PIO.

Genauer gesagt man beeinflußt das Ausgaberegister von Port C. Will man ein einzelnes Bit im Ausgaberegister von Port C setzen oder löschen, so geschieht dies auf folgende Art und Weise:

Bits:	7	6	5	4	3	2	1	0
Bedeutung:	0	0	0	0	x	x	x	y

Mit den Bits 3, 2 und 1 wird eines der 8 Bits von Port C bestimmt. Bit 0 gibt nun an, ob dieses Bit gesetzt oder gelöscht werden soll.

Will man beispielsweise den Kassettenmotor (Port C, Bit 4) einschalten, dann geht man folgendermaßen vor:

Der Motor soll eingeschaltet werden, also setzt man Bit 0 des Befehls-Bytes auf 1. Zur Motorsteuerung wird das Bit 4 verwendet. Im Binärsystem hat die Zahl Vier folgende Schreibweise: %00000100

Die Zahl 4 muß nun noch um ein Bit nach links verschoben werden, da ja im Bit Null des Befehls-Bytes das Ein-/Aus-Bit steht.

Es wird also Bit 3 des Steuerregisters auf Eins, und die Bits 2 und 1 auf Null gesetzt. Das Steuerwort hat demnach folgenden Aufbau:

```
! Bits:   7 ! 6 ! 5 ! 4 ! 3 ! 2 ! 1 !
!-----------+---+---+---+---+---+---+
! Inhalt: 0 ! 0 ! 0 ! 1 ! 0 ! 0 ! 1 !
```

Durch diese Spezialität der PIO ist es möglich das Ausgaberegister von Port C bitweise zu beeinflussen, ohne sich um die andern Bits kümmern zu müssen. Durch diese Methode lassen sich aber wie gesagt nur die Bits im Ausgaberegister beeinflussen.

Der 6845 CRTC Video-Prozessor des CPC

Die Fähigkeiten dieses überaus komplexen Chips seinen hier nur kurz angesprochen. Der CRTC hat 18 Register, er wird über vier Portadressen gesteuert.

&BC?? : Selektiert Register des CRTC, das bearbeitet werden soll.
&BD?? : Bekommt Werte, die in ein CRTC Register geschrieben werden sollen.
&BF?? : Dient dazu Werte vom CRTC zu lesen

Es folgen einige Infos zu den 18 Registern des CRTC:

Regster 0 - Horizontal Total : alle Zeichen pro Zeile - 1 inclusive dem Rand. Einfluß auf CLK Takte, und damit auf Start des H-Sync.

Regster 1 - Horizontal Displayed : Zahl angezeigter Zeichen pro Zeile.

Regster 2 - Horizontal Sync Position : Position des H-Sync Signals in jeder Rasterzeile (in CLK Takten). ==> Bild horizontal verschiebbar.

Regster 3 - Horizontal Sync Width : Länge des H-Sync in CLK Takten.

Regster 4 - Vertical Total : Zahl der Zeichenzeilen incl. Rand.

Regster 5 - Vertical Total Adjust : Feineinstellung (Rasterzeilen) von Reg.4.

Regster 6 - Vertical Displayed : Anzahl angezeigter Zeichenzeilen.

Regster 7 - Vertical Sync Position : Position des V-Sync Signals in Zeilen. Bild auf/ab verschiebbar.

Regster 8 - Interlace Mode and Scew : Interlace-Steuerung

Regster 9 - Maximum Scan Line Address : Höhe eines Zeichens in Rasterzeilen - 1.

Regster 10/11 - Cursor Start/Cursor End : Start- & End-Rasterzeile des Hardware-Cursors.

Regster 12/13 - Start Address : StartAd. des Bildschirm-RAMs, Overscan aktivieren. 12 ist ein 6 Bit Register. Bei den Plus CPCs lassen sich beide lesen & schreiben, bei den CPCs old Gen. nur schreiben!

Regster 14/15 - Cursor Register : Cursor-Adresse setzen oder lesen.

Regster 16/17 - Light Pen : Adr. des LPSTRB Signals lesbar.

Da diese Datei bereits lang genug ist, verzichte ich momentan darauf den FDC765, den PSG AY-3-8912 und die MM-Baugruppen der Plus CPCs zu dokumentieren. Habe ich mal etwas Zeit, dann wird diese Datei aktualisiert.

Die aktuellste Version dieser Datei (Prowort, Word oder ASCII Format) kann im Internet runtergeladen werden:

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