

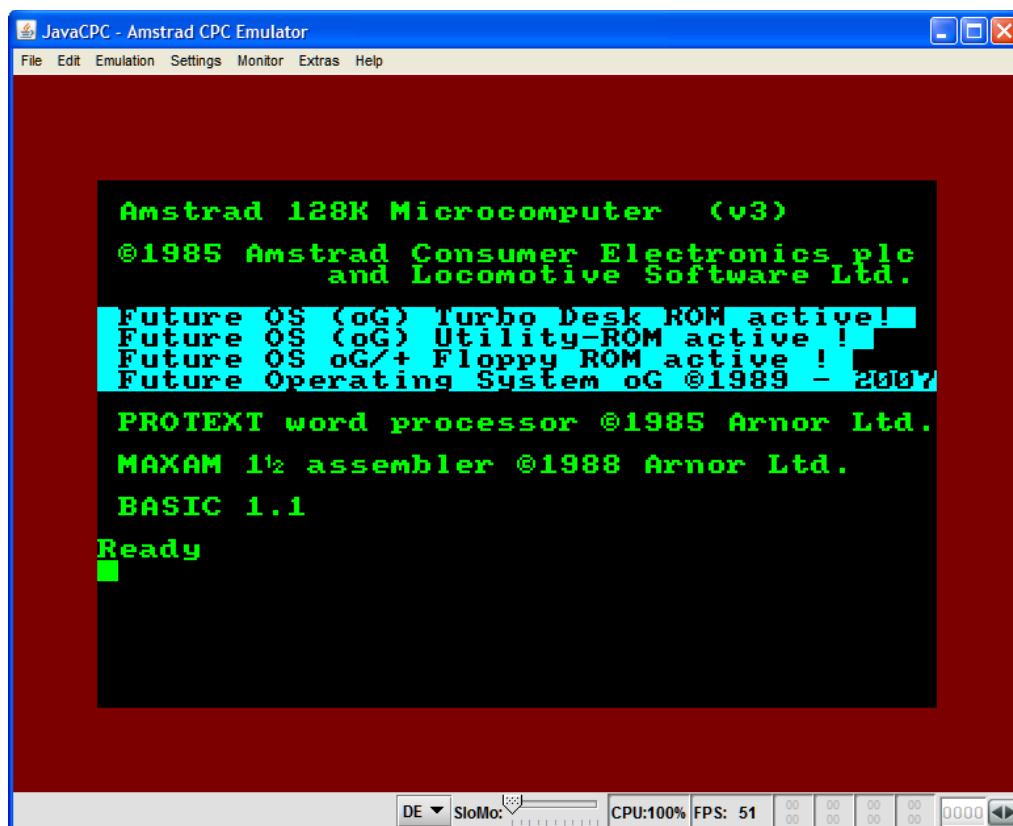
Programmieren unter dem Betriebssystem FutureOS in Assembler für Anfänger, Fortgeschrittene und Profis

Voraussetzungen um ein Assembler Programm für FutureOS zu erstellen

- Ein Amstrad / Schneider Computer **CPC6128** bzw. ein **6128 Plus** mit Grünmonitor GT65 (empfohlen) oder Farbmonitor CTM644 ist vorhanden.
- Das Betriebssystem **FutureOS** (System-Version .8 oder höher) ist in einer EPROM-Karte, dem MegaFlash, der ROM-RAM-Box, dem SYMBiFACE II oder III, dem X-MEM, dem M4 oder einer ähnlichen Karte installiert. FutureOS kann unter <http://www.FutureOS.de> geladen werden.
- Ein **Assembler** steht zur Verfügung. Die Beispiele in diesem Handbuch beziehen sich auf den MAXAM Assembler im ROM, können aber auf jeden Z80 Assembler übertragen werden, der in der Lage ist Binär-Dateien (*.BIN) für den CPC zu schreiben.
Das ROM des MAXAM Assemblers kann unter <ftp://ftp.nvg.ntnu.no/pub/cpc/emulator/rom/> heruntergeladen werden (Dort findet man ebenfalls das Protext ROM falls man MAXAM 1½ benutzen möchte).
- Grundlegende Kenntnisse über **Maschinensprache (Z80)** sollten vorhanden sein.

Anmerkungen: Anstatt des **MAXAM 1.15** Assembler ROMs können auch zwei ROMs mit **Protext** und **MAXAM 1½** installiert werden, hierbei dient Protext als Texteditor.

Die Einschaltmeldung des CPCs bzw. des CPC Emulators sollte in etwa so aussehen:



Sollte kein echter CPC6128 zur Verfügung stehen, so kann auch ein CPC Emulator verwendet werden, Beispiele wären Caprice, JavaCPC, WinApe, WinCPC etc.

Vorbereitungen zum Schreiben eines Programms

Sie benötigen eine Diskette mit der Datei **"#EQU-API.DEU"**. Diese Datei enthält alle EQUates, also alle Address-Zuweisungen für System-Variablen und System-Funktionen des Applikations-Programm-Interface (API).

Wenn sie auf Variablen des Systems zugreifen wollen, oder Funktionen des Betriebssystems aufrufen wollen, dann benötigen Sie deren Adresse im Speicher. Und genau diese Adressen sind in der Datei **"#EQU-API.DEU"** angegeben. Ein DSK Image welches diese Datei enthält kann hier heruntergeladen werden: http://www.colorado-boys-muenchen.de/users/futureos/files/Variable_Definitions_and_EQUates.zip

Sollen Variablen des Betriebssystems (hier: FutureOS) oder Bezeichnungen von Funktionsaufrufen (sogenannte Labels) verwendet werden, so ist am Anfang der Zeile das Semikolon zu entfernen.

Es gibt drei Möglichkeiten um mit der Variablen- und Funktions-Definitionsdatei **"#EQU-API.DEU"** zu arbeiten:

- Die gesamte Datei **"#EQU-API.DEU"** wird in den Quellcode des Programms integriert. Das Semikolon am Zeilenanfang wird bei allen benötigten Variablen und Funktionsaufrufen gelöscht.
- Die Datei **"#EQU-API.DEU"** wird mittels der Anweisung: **READ"#EQU-API.DEU"** vom Quellcode des Programms eingelesen. Dabei wird das Semikolon am Zeilenanfang bei allen benötigten Variablen und Funktionsaufrufen gelöscht.
(Mittels des Befehls **READ"Datei"** ist es dem MAXAM Assembler möglich zusätzliche Dateien an dieser Stelle in den Source-Code einzulesen und zu integrieren. READ kann aber nur im Hauptprogramm verwendet werden, eine Verschachtelung ist nicht möglich.)
- Die vom Programm / Quellcode benötigten Variablen-Bezeichnungen und die Namen der Funktionsaufrufe werden aus der Datei **"#EQU-API.DEU"** in den Quellcode kopiert. Das Semikolon am Anfang einer jeden Zeile wird entfernt.

Mein erstes Programm „Hallo Welt“

Den Quell-Code eingeben

Um den Quell-Code eines Programms einzugeben ist folgendes zu tun:

1. Bei Verwendung von MAXAM 1.15

- a. Schalten Sie den CPC ein (MAXAM und FutureOS vorhanden)
- b. Legen Sie eine leere Diskette ins Laufwerk ein
- c. Springen Sie in den MAXAM Assembler durch Eingabe des Kommandos **|m,2** (dabei schaltet der Parameter „2“ nach dem Komma in den Bildschirm MODE 2).
- d. Sie befinden sich nun im Monitor-Teil des MAXAM Assemblers, bitte geben Sie nun den Buchstaben **„T“** gefolgt von Return ein, um in den Text-Editor zu gelangen.
- e. Nun befinden Sie sich im Menu des Text-Editors. Bitte geben Sie **„E“** ein, um Text eingeben zu können.
- f. Im Text-Editor angelangt können Sie nun den Quell-Code eines Programms eingeben.
- g. >> Jetzt bitte den Quell-Code eingeben << Nach Eingabe des Quell-Codes kann der Editor durch drücken der **ESC** Taste verlassen werden.
- h. Mit **„S“** können Sie nun ihren Quell-Code auf Diskette abspeichern.

2. Bei Verwendung von MAXAM 1½ und Protex

- a. Schalten Sie den CPC ein (MAXAM 1½, Protext und FutureOS vorhanden)
- b. Legen Sie eine leere Diskette ins Laufwerk ein
- c. Springen Sie in den Protext Text-Editor durch Eingabe des Kommandos **|p**.
- d. Sie befinden sich nun im Befehls-Modus von Protext, bitte geben Sie nun den Befehl „prog“ gefolgt von Return ein, um den Text-Editor auf Quell-Code einzustellen.
- e. Nun blinkt der Cursor im Text-Editor. Hier können Sie nun den Quell-Code ihres Programms eingeben.
- f. >> Jetzt bitte den Quell-Code eingeben << Nach Eingabe des Quell-Codes kann der Editor durch drücken der **ESC** Taste verlassen werden.
- g. Durch Eingabe des Befehls „s“ und eines gültigen Dateinamens können Sie nun ihren Quell-Code auf Diskette abspeichern.

Anmerkung: Alle mit / für MAXAM erstellten Quell-Codes sollten immer die Datei-Erweiterung „**MAX**“ enthalten. So lässt sich leicht erkennen, dass es sich um Quell-Code für MAXAM handelt.

Nun wollen wir gleich ein einfaches „Hallo Welt“ Programm eingeben, das einfach nur „Hallo Welt“ auf den Bildschirm schreibt. Den Quell-Code finden Sie auf der nächsten Seite. Bitte geben Sie den Quell-Code in den Assembler bzw. mit Protext ein, und vergessen Sie nicht den Quell-Code auf Diskette unter dem Namen „hello.max“ abzuspeichern. (Falls Sie mit diesem Handbuch eine Diskette erhalten haben, so befindet sich der Quell-Code bereits fertig eingetippt auf der Diskette, sein Name ist „HELLO.MAX“).

```

;Hello World - Name dieser Datei HELLO.MAX
;
;Dieses Programm schreibt "Hello World" in den unteren Bereich des FutureOS
;Desktop. Dabei werden die Icons nicht verändert.

ORG &8000      ;Der Befehl ORG legt die Adresse fest, ab welcher der
               ;assemblierte Maschinen-Code im Speicher abgelegt werden soll.
               ;In diesem Fall wird der Maschinen-Code ab &8000 abgelegt.

WRITE"HW.64K"  ;Der Befehl WRITE schreibt den generierten Maschinen-Code
               ;auf Diskette. Dabei wird der Dateinamen "HW.64K"
               ;verwendet.
               ;Die Erweiterung „.64K“ zeigt an, dass das assemblierte
               ;Programm im Hauptspeicher ausgeführt wird.
               ;Das assemblierte Binär-Programm „HW.64K“ bekommt vom
               ;Assembler automatisch eine Autostart-Adresse zugewiesen.
               ;Die Autostart-Adresse entspricht hierbei der Adresse,
               ;ab dem der Code abgelegt wird, siehe ORG Befehl.
               ;In diesem Falle ist das wiederum &8000.
               ;Ein so abgespeichertes Programm kann deshalb direkt
               ;unter FutureOS gestartet werden.
               ;
               ;Ohne den Befehl WRITE würde der assemblierte Code direkt
               ;im Speicher abgelegt werden. In diesem Falle ab &8000.

;Ab hier beginnt der eigentliche Programm-Code

HELLO LD BC, (&FF01)  ;Das Doppelregister BC wird aus der Adresse &FF01
                     ;geladen. Diese 16 Bit Adresse enthält die Nummer
                     ;von ROM A des FutureOS (in unteren Byte) und den
                     ;Wert &DF (im oberen Byte).
                     ;Über den Port &DFxx wird im CPC das obere ROM
                     ;ausgewählt. Normalerweise liegt ROM A des
                     ;FutureOS auf ROM-Auswahl 10 (&0A in hexadezimal)
                     ;In diesem Fall enthält BC den Wert &DF0A

OUT (C),C           ;Dieser Befehl sendet den Wert des Registers C
                     ;an die durch BC adressierte Ausgabe-Schnittstelle.
                     ;Also wird &0A an den Port &DF0A gesendet.
                     ;&DFxx ist für die obere ROM Auswahl zuständig.
                     ;Es wird also FutureOS ROM A eingeblendet.
                     ;Dieses obere ROM liegt zwischen &C000 und &FFFF.
                     ;
                     ;Das FutureOS ROM A wurde ausgewählt, um darin
                     ;enthaltene Funktionen ausführen zu können.

LD BC,&7F82          ;Die Ausgabeschnittstelle &7Fxx ist für die Auswahl
                     ;von RAM, ROM und des Bildschirm-MODEs zuständig.
                     ;Der Wert &82 bewirkt dabei folgendes;
                     ;- Das obere ROM wird eingeblendet, anstatt RAM
                     ;   (Damit wird das FutureOS ROM A eingeblendet
                     ;   Funktionen in OS ROM können so benutzt werden)
                     ;- Das untere ROM wird zwischen &0000 und &3FFF
                     ;   eingeblendet. (Und damit wird auch der normale
                     ;   Zeichensatz des CPC von &3800 bis &3FFF
                     ;   eingeblendet. Das ist nötig, da sich im RAM
                     ;   bisher noch kein Zeichensatz befindet)
                     ;- Der Bildschirm-Mode wird auf 2 belassen
                     ;   (MODE 2 war auch zuvor schon eingeblendet)

OUT (C),C           ;Wert &82 an Port &7Fxx senden, damit ROMs ein.

```



```
LD (RAMCHAR),BC ;Der aktuelle MODE und der Status des oberen und
;des unteren ROMs wird in die RAM Variable
;RAMCHAR geschrieben.
;Dadurch wird dem OS der MODE und der RAM Status
;mitgeteilt.
```

```
;Natürlich ließe sich das auch in Form einer OS
;Funktion realisieren, das wäre aber deutlich
;umständlicher, und vor allem langsamer, als die
;Verwendung von drei Z80 Befehlen
```

```
;Bisher wurde das FutureOS ROM A ab &C000 eingeblendet. Und es wurde
;das untere ROM mit dem darin enthaltenen Zeichensatz eingeblendet.
```

```
LD HL,TXT ;Das Doppelregister HL wird mit der Adresse des
;auszugebenden Textes geladen. Diese 16 Bit Adresse
;wird auch als Zeiger, Vektor oder Pointer etc.
;bezeichnet, da sie auf ein LABEL (eine Marke)
;im RAM des CPC zeigt.
```

```
CALL TERM_2 ;FutureOS stellt mehrere OS-Funktionen zur Ausgabe
;von Text auf dem Bildschirm zur Verfügung.
;Mit TERM_2 können normale Zeichen im MODE 2
;ausgegeben werden.
;Beim Aufruf vom TERM_2 muss das Doppelregister HL
;mit einer Adresse geladen sein, die auf den Text
;zeigt, der dargestellt werden soll.
```

```
;Der Text "Hello World!" wurde nun auf dem Bildschirm ausgegeben!
```

```
;Am Ende des Programms muss das Programm die Kontrolle wieder an das OS
;zurückgeben. Dies geschieht z.B. auf folgende Weise..
```

```
LD HL,KLICK ;Die OS Funktion KLICK ist ein Einsprung ins OS.
;Sie wird nur dann verwendet, wenn das ausführende
;Programm die Icons in der oberen Haelfte des
;Desktop nicht verändert hat.
;In unserem Falle hat das Programm lediglich im
;unteren Bereich des Bildschirms Text ausgegeben.
;Die Systemfunktion KLICK kann also verwendet
;werden.
;Das sich KLICK aber im ROM D befindet, wir aber
;ROM A eingeblendet haben, kann KLICK nicht direkt
;aufgerufen werden. Stattdessen wird das Doppelreg.
;HL mit der Adresse von KLICK geladen. Und...
```

```
JP ROM_D ;Die Systemfunktion ROM_D ist in allen vier ROMs
;des FutureOS vorhanden, sie kann also immer
;aufgerufen werden.
;Das Programm springt hier direkt zu ROM_D.
;Die Systemfunktion ROM_D blendet das FutureOS
;ROM D ein und springt anschließend an die Adresse,
;die in HL angegeben ist. In diesem Beispiel ist
;das die Adresse der Systemfunktion KLICK.
```

```
;Anstelle der letzten beiden Befehle wäre es auch möglich zuerst FutureOS
;ROM D einzublenden und dann direkt zu KLICK zu springen.
```

```
;
```

```
;Das würde dann so aussehen:
```

```
;LD BC,(&FF13) ;Schnittstelle ROM-select und Nummer von ROM D in BC laden
;OUT (C),C ;FutureOS ROM D auswählen
;JP KLICK ;Zurück ins OS springen
```



```
;Auszugebender Text ab dem LABEL TXT
;Der auszugebende Text besteht aus einem Steuerzeichen (drei Bytes),
;die den Cursor auf dem Bildschirm positionieren, und aus dem Text
;"Hello World".
```

```
TXT    DB &1E,21,26    ;Der Text beginnt mit einem Steuerzeichen, also mit
                        ;einem sogenannten Control Code.
                        ;Der Control Code &1E arbeitet ähnlich wie der
                        ;BASIC Befehl LOCATE, er positioniert den Cursor
                        ;auf dem Bildschirm. Dabei wird vorausgesetzt, dass
                        ;sich der Bildschirm im 64 x 32 Modus befindet.
                        ;Das bedeutet 64 Zeichen pro Zeile, und 32 Zeilen.
                        ;Beim Aufruf eines Programms befindet sich der
                        ;Bildschirm bereits in diesem Mode, er muss also
                        ;nicht extra eingestellt werden.
                        ;Dem Control Code &1E (LOCATE) folgen zwei Bytes,
                        ;die die Y und X Position angeben, auf die der
                        ;Cursor gesetzt werden soll.
                        ;Die Y-Position ist hierbei die 21. Zeile.
                        ;Und X-Position ist die 26. Spalte.
```

```
DB "Hello World!" ;Dies ist der auszugebende Text
```

```
DB 0                ;Der Text wird durch ein Null-Byte beendet.
                    ;Neben Control-Code &00 kann auch der Control-Code
                    ;&1A als String-Terminator verwendet werden
```

```
;Die Steuerzeichen des FutureOS sind mit ausführlicher Beschreibung im
;großen Handbuch zu finden.
```

```
;Nachfolgend werden die LABELs der System-Variablen und System-Funktionen
;aufgelistet.
;Diese EQUates sind direkt von der Datei "#E.D" übernommen.
```

```
;EQUates of OS variables
```

```
RAMCHAR EQU &B847 ;located in first 64 KB RAM
```

```
;EQUates of System Calls
```

```
TERM_2 EQU &D48C    ;located in ROM A (Textausgabe)
KLICK EQU &FE9A      ;located in ROM D (Rücksprung in den Desktop)
ROM_D EQU &FF12      ;located in all ROMs
```

Den Quellcode abspeichern

In jeden Fall sollte der frisch eingeebene oder veränderte Quellcode sofort auf Diskette abgespeichert werden. Es empfiehlt sich hierbei auf mehrere Disketten zu speichern, die an verschiedenen Orten aufbewahrt werden.

Bei der Verwendung eines Emulators sollten die generierten DSK Images auf verschiedenen Festplatten gesichert werden. Es empfiehlt sich auch regelmäßige Sicherheitskopien auf CD bzw. USB Massenspeicher zu ziehen.

Das Assemblieren des Programms

Nach der Eingabe bzw. dem Bearbeiten des Quell-Codes – und nachdem Sie ihn auf Diskette gesichert haben – kann der Quell-Code nun assembliert werden. Assemblieren bedeutet, dass der Assembler den Quell-Code in ein Binär-Programm übersetzt, so ein Binär-Programm kann vom Z80 Prozessor direkt ausgeführt werden.

Das Assemblieren eines Programms wird folgendermaßen durchgeführt:

1. Bei Verwendung von MAXAM 1.14

- a. Der Quell-Code wurde entweder gerade unter MAXAM eingegeben, oder im Text-Editor-Menü von MAXAM mittels des Kommandos „L“ von Diskette geladen.
- b. Legen Sie nun bitte die Diskette ins Laufwerk, auf die das fertige Programm abgespeichert werden soll.
- c. Im Text-Editor-Menü des MAXAM Assemblers geben Sie nun den Befehl „A“ ein, gefolgt von Return. MAXAM beginnt nun zu assemblieren.
- d. Nach dem Ende der Assemblierung drücken Sie bitte eine Taste um ins Text-Editor Menü von MAXAM zurückzukehren.
- e. Das generierte Binär-Programm befindet sich nun auf Diskette
- f. Bei diesem Beispiel trägt es den Datei-Namen „HW.64K“

2. Bei Verwendung von MAXAM 1½ und Protext

- a. Der Quell-Code wurde entweder gerade unter Prowort eingegeben, oder im Befehls-Modus von Prowort mittels des Kommandos „L“ von Diskette geladen.
- b. Legen Sie nun bitte die Diskette ins Laufwerk, auf die das fertige Programm abgespeichert werden soll.
- c. Im Befehls-Modus von Prowort geben Sie nun den Befehl „asm“ ein, gefolgt von Return. Dadurch wird der MAXAM Assembler aufgerufen und beginnt nun zu assemblieren.
- d. Nach dem Ende der Assemblierung drücken Sie bitte eine Taste um in den Befehls-Modus von Prowort zurückzukehren.
- e. Das generierte Binär-Programm befindet sich nun auf Diskette
- f. Bei diesem Beispiel trägt es den Datei-Namen „HW.64K“

Anmerkung: Um das Inhaltsverzeichnis der eingelegten Diskette anzuzeigen können Sie unter MAXAM mit „X“ in den RSX Modus wechseln, dort geben sie „cat“ ein, wie gewohnt. Das Inhaltsverzeichnis der Diskette wird nun angezeigt. Drücken Sie anschließend **ESC** um ins vorige Menü zurückzukehren. Unter Prowort geben Sie einfach „cat“ im Befehls-Modus ein um das Inhaltsverzeichnis der gerade eingelegten Diskette anzuzeigen.

Ausführen eines Programms unter FutureOS

Bisher haben Sie den Quell-Code eines Programms in den Text-Editor eingegeben, Sie haben den Quell-Code auf Diskette gespeichert und anschließend haben Sie bereits den Quell-Code assembliert. Das assemblierte Binär-Programm befindet sich auf Diskette, und nun wollen wir es unter FutureOS starten.

Starten eines Programms unter FutureOS

- Den CPC6128 oder 6128 Plus einschalten. FutureOS zeigt seine Initialisierungs-Meldung
- Mit dem Befehl „|OS“ das Betriebssystem FutureOS booten
- Die Diskette mit dem zu startenden Programm in ein Laufwerk einlegen
- Das entsprechende Laufwerks-Icon anklicken. Dabei symbolisiert das Icon „**A**“ das interne 3“ Laufwerk des CPC, und das Icon „**B**“ symbolisiert das am Disc-Drive-B Anschluss befindliche Laufwerk (egal ob 3“, 3.5“ oder 5.25“)
- Nun klicken Sie bitte das „**DIR**“ Icon an. Nach dem Anklicken sehen Sie im unteren Teil des Bildschirms das Inhaltsverzeichnis des gewählten Laufwerks
- **Anmerkung:** Sollten Sie mehrere Laufwerke markiert haben, oder sollte die Diskette mehr als 64 Dateien beinhalten, so lässt sich mittels der Tasten „**Control**“ und „**Shift**“ durch das Inhaltsverzeichnis blättern
- In dem angezeigten Inhaltsverzeichnis klicken Sie nun bitte das zu startende Programm an
- Anschließend klicken Sie das „**RUN**“ Icon an
- Nun wird das Programm geladen und ausgeführt
- Nach dem Ende des Programms können Sie FutureOS mittels des „**END**“ Icons verlassen

Ein etwas komplexeres Programm

Die Zusammenarbeit mit dem Desktop

Programme können intensiv mit dem Desktop des FutureOS verknüpft werden. Auf diese Weise muss sich das Programm nicht um eine eigene Oberfläche kümmern, sondern bedient sich ganz einfach der „Turbo Desktop“ Oberfläche des Betriebssystems.

Da die Icons normalerweise auf fixen Positionen angeordnet sind wird sich der Anwender auch auf jedem CPC mit FutureOS zurechtfinden.

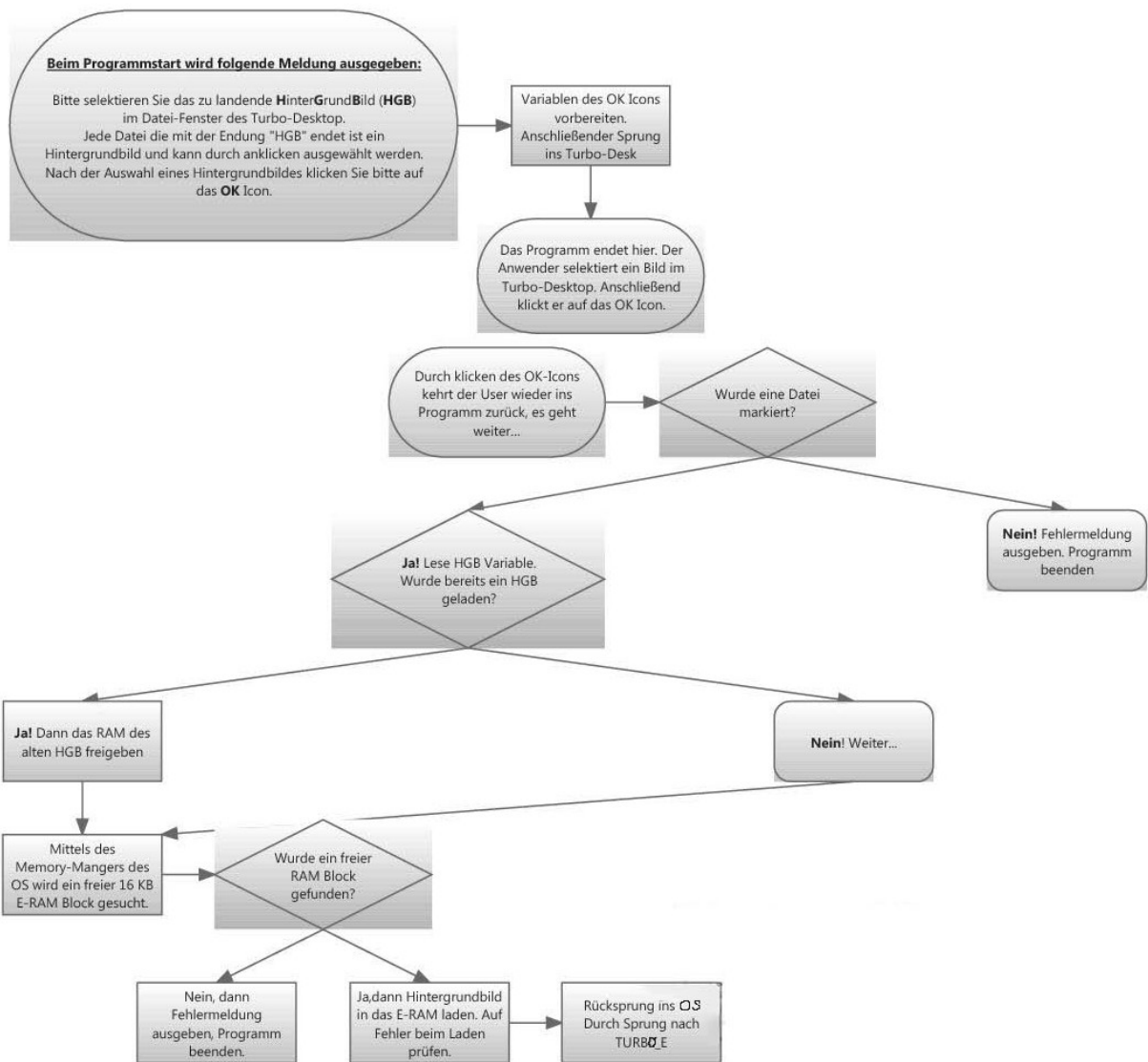
Im folgenden Beispiel wird soll ein Programm entwickelt werden, welches möglichst viele Funktionen des OS und des Desktop benutzt.

Beispielprogramm: Auswahl und Laden eines Hintergrundbildes

Sinn dieses einfachen Programms ist es ein neues Hintergrundbild von Diskette oder Festplatte zu laden und zu aktivieren.

Zuerst wollen wir hier ein Syntax-Diagramm entwickeln, zuerst in Text dann in Diagrammform. Anschließend werden wir es Stück für Stück in Programm-Quellcode übersetzen.

- Meldung in der unteren Bildschirmhälfte: „Bitte selektieren Sie das zu landende **HinterGrundBild** (HGB) im Datei-Fenster des Turbo-Desktop. Jede Datei die mit der Endung „HGB“ endet ist ein Hintergrundbild und kann durch anklicken ausgewählt werden. Nach der Auswahl eines Hintergrundbildes klicken Sie bitte auf das **OK** Icon.“
- Variablen des OK Icons vorbereiten
- Sprung ins Turbo-Desktop. Verwendung vom SCRI als Einsprung
- --- nun selektiert der User im Desktop eine Datei...
- Durch klicken des OK-Icons kehrt der User wieder ins Programm zurück, es geht weiter...
- Überprüfen, ob überhaupt eine Datei markiert wurde
 - o Nein, dann eine Fehlermeldung ausgeben
 - o Ja, dann weiter im Programm
- Prüfen, ob bereits ein Hintergrundbild geladen wurde
 - o Ja, dann das entsprechende RAM freigeben. Und weiter im Programm
 - o Nein, dann weiter im Programm
- Mittels OS Funktion das oberste freie RAM suchen. Test ob RAM gefunden
 - o Nein, dann Fehlermeldung und Programm abbrechen
 - o Ja, dann weiter im Programm
- Die Variable des freien Erweiterungs-RAMs (E-RAM) anpassen. Dadurch das OS informieren.
- Das Hintergrundbild wird nun in das entsprechende E-RAM geladen
- Prüfen, ob beim Laden des Bildes ein Fehler auftrat
 - o Ja, dann RAM freigeben und Programm abbrechen
 - o Nein, dann ...
- Programm beenden, durch Sprung in das OS mittels Einsprung TURBO_E




```

LD HL,KLICK ;Die OS Funktion KCLICK ist ein Einsprung ins OS.
              ;Das sich KCLICK in ROM D befindet, aber ROM A eingeblendet ist,
              ;kann KCLICK nicht direkt aufgerufen werden.
              ;Deshalb wird Doppelregister HL mit der Adresse von KCLICK geladen

JP ROM_D ;Die Systemfunktion ROM_D ist in allen vier ROMs des FutureOS
          ;an derselben Stelle vorhanden, man kann sie also immer aufrufen
          ;Die Systemfunktion ROM_D blendet zuerst das FutureOS ROM D ein und
          ;springt anschließend an die Adresse, die in HL angegeben ist.

;-----;
;Der Anwender klickt nun ein HintergrundBild (HGB) seiner Wahl an.
;Anschließend klickt er auf das OK-Icon.
;Das OK Icon liest die OK Variablen ein, es blendet die entsprechende RAM-
;Konfiguration ein und springt an die angegebene Zieladresse...
;-----;

;Die Variablen des OK-Icons wurden zuvor so gesetzt, dass das OS an das LABEL
;"HGB_WEITER" springt, sobald das OK-Icon angeklickt wird.

HGB_WEITER LD A,&FF ;Parameter f. OS-Funktion FMD32 in ROM C. Sorgt dafür,
                  ;dass die erste gefundene Datei markiert bleibt.

LD IX,FMD32 ;Register IX wird mit der Adresse der Funktion FMD32
            ;geladen. FMD32 findet die erste markierte Datei.

CALL ROM_A2C ;Die Funktion ROM_A2C ruft eine OS Funktion in ROM C auf,
            ;deren Adresse muss sich in Register IX befinden.

LD A,L ; / FMD32 schreibt die Adresse der ersten markierten Datei in HL.
OR A,H ; / Wenn das Register HL aber den Wert &0000 enthält, dann wurde
        ;/ keine markierte Datei gefunden. Hier werden H und L addiert.

JR Z,NO_FILE ;Ist das Ergebnis dieser Addition &00, dann wurde keine Datei
            ;markiert. In dem Fall wird zur Fehler-Ausgabe gesprungen.

LD A,(HGB_ST) ;Der Akku (Register A) wird mit dem Wert aus der OS Variable
              ;HGB_ST geladen. Ist diese Variable = &FF dann wurde bereits
              ;ein HintergrundBild (HGB) geladen.
              ;Das Programm testet das, um nicht unnötig RAM zu belegen.

INC A:JR NZ,NO_GHB ;Wenn bisher noch kein HGB geladen ist, so wird ein
                  ;Teil des Programms übersprungen, bis zum LABEL NO_GHB

LD A,(HGB_RB) ;Erweiterungs-RAM (physikalischer I/O Wert) in Register A lesen

CP A,&C0:JR Z,NO_GHB ;War das alte HGB im Hauptspeicher, dann unter weiter...

LD B,A ;Register A in B sichern

AND A,&03 ;Die unteren 2 Bits (0000 00XX) von A isolieren...

LD C,A ;und in Register C sichern.

LD A,B ;E-RAM Konfiguration aus C in A holen

AND A,&38 ;Isoliere die mittleren 3 Bits (00BB B000)

RRCA ;Rotieren alles Bits um Eins nach rechts (000B BB00)

OR A,C ;Und addiere die unteren beiden Bits (000B BXX)

ADD A,&D0 ;Addiere das untere Byte der Adresse der RAM-Variablen (XRAM_C4)

LD L,A ; / Register HL zeigt nun auf eine von 32 RAM Variablen, ab Adresse
LD H,&B9 ;/ &B9D0 (bis max. &B9EF). XRAM_C4 = &B9D0

LD (HL),&01 ;In die E-RAM Variable des alten E-RAMs des alten HGB wird der
            ;Wert &01 geschrieben, dadurch wird es freigegeben.

```

```

NO_GHB CALL EFER ;Aufruf des Memory-Mangers. Diese Funktion sucht nach einem
                  ;freien 16 KB Erweiterungs-RAM Block. Die Suche beginnt oben.

JR Z,NO_RAM ;Ist das Z-Flag gesetzt, dann wurde kein RAM gefunden -> Fehler!

LD (HGB_RB),A ;Physikalisches E-RAM für das HGB buchen, für das OS
               ;FutureOS holt das HGB aus diesem E-RAM um es anzuzeigen

LD (AKT_RAM),A ;E-RAM auf für das Laden des Bildes in die entsprechende
               ;RAM Variable des OS schreiben. Siehe unten...

LD A,&FF
LD (HGB_ST),A ;Der Wert &FF teilt dem OS mit, dass ein HGB vorhanden ist

;Nun kann das HGB geladen werden...

LD A,&03 ;Register A mit &03 laden, Typ 3 - Laden lädt in das E-RAM
LD (REG08_3),A ;Und Lade Typ 3 in RAM Variable REG08_3 sichern

LD HL,&4000 ;Die Zieladresse ist &4000 im E-RAM Block, siehe oben, AKT_RAM
LD (REG16_1),HL ;Und die Zieladresse in RAM Variable REG16_1 speichern

LD IX,LADEN ;Das Register wird mit der Adresse der OS Funktion LADEN geladen
CALL ROM_A2C ;Und mittels der OS Funktion ROM_A2C wird die in IX gespeicherte
             ;Adresse / Funktion aufgerufen. Anschließend ROM A einblenden.

;HIER BITTE NOCH PRÜFEN, OB DATEI KORREKT GELADEN WURDE!!!
;Wenn XXX, ist dann ist ein Fehler aufgetreten --> Sprung nach LD_ERR

;Am Ende des Programms muss das Programm die Kontrolle wieder an das OS
;zurückgeben. Dies geschieht z.B. auf folgende Weise...

LD HL,TUR_E ;Die OS Funktion TUR_E ist ein Einsprung ins OS.
             ;Das sich TUR_E in ROM D befindet, aber ROM A eingeblendet ist,
             ;kann TUR_E nicht direkt aufgerufen werden.
             ;Deshalb wird Doppelregister HL mit der Adresse von TUR_E geladen

JP ROM_D ;Die Systemfunktion ROM_D ist in allen vier ROMs des FutureOS
          ;an derselben Stelle vorhanden, man kann sie also immer aufrufen
          ;Die Systemfunktion ROM_D blendet zuerst das FutureOS ROM D ein und
          ;springt anschließend an die Adresse, die in HL angegeben ist.

;Dieser Teil des Programms wird nur dann ausgeführt, wenn der Anwender kein
;Bild markiert hat. In diesem Fall wird eine Fehlermeldung ausgegeben.

NO_FILE LD BC,&7F82 ;Das untere ROM (und damit der Zeichensatz) wird...

OUT (C),C ;eingeblendet. (Auch das obere ROM, und MODE bleibt 2).

LD HL,NOFILE ;HL mit der Adresse des Textes der Fehlermeldung laden

CALL TERM_2I ;Mit TERM_2I werden invertierte Zeichen im MODE 2 ausgegeben
             ;Beim Aufruf vom TERM_2I muss das Doppelregister HL mit einer
             ;Adresse geladen sein, die auf den darzustellenden Text zeigt

LD HL,KLICK ;Mittels der OS Funktion KLICK wird das Programm beendet

JP ROM_D ;Die Systemfunktion ROM_D blendet zuerst das FutureOS ROM D ein und
          ;springt anschließend an die Adresse, die in HL angegeben ist (KLICK)

```

;Dieser Teil des Programms wird nur dann ausgeführt, wenn kein freies
;Erweiterungs-RAM vorhanden ist. Also wird eine Fehlermeldung ausgegeben.

NO_RAM LD BC,&7F82 ;Das untere ROM (und damit der Zeichensatz) wird...

OUT (C),C ;eingebledet. (Auch das obere ROM, und MODE bleibt 2).

LD HL,NOERAM ;HL mit der Adresse des Textes der Fehlermeldung laden

CALL TERM_2I ;Mit TERM_2I werden invertierte Zeichen im MODE 2 ausgegeben
;Beim Aufruf vom TERM_2I muss das Doppelregister HL mit einer
;Adresse geladen sein, die auf den darzustellenden Text zeigt

LD HL,KLICK ;Mittels der OS Funktion KLICK wird das Programm beendet

JP ROM_D ;Die Systemfunktion ROM_D blendet zuerst das FutureOS ROM D ein und
;springt anschließend an die Adresse, die in HL angegeben ist (KLICK)

;Dieser Teil des Programms wird nur dann ausgeführt, wenn beim LADEN
;des HinterGrundBildes (HGB) ein Fehler aufgetreten ist. Fehlermeldung!

LD_ERR LD BC,&7F82 ;Das untere ROM (und damit der Zeichensatz) wird...

OUT (C),C ;eingebledet. (Auch das obere ROM, und MODE bleibt 2).

LD HL,LADERR ;HL mit der Adresse des Textes der Fehlermeldung laden

CALL TERM_2I ;Mit TERM_2I werden invertierte Zeichen im MODE 2 ausgegeben
;Beim Aufruf vom TERM_2I muss das Doppelregister HL mit einer
;Adresse geladen sein, die auf den darzustellenden Text zeigt

XOR A,A ;Den Akku leeren, d.h. Register A = &00

LD (HGB_ST),A ;Der Wert &00 teilt dem OS mit, dass KEIN HGB vorhanden ist
;Das ist nötig, da beim LADEN ein Fehler aufgetreten ist

LD HL,KLICK ;Mittels der OS Funktion KLICK wird das Programm beendet

JP ROM_D ;Die Systemfunktion ROM_D blendet zuerst das FutureOS ROM D ein und
;springt anschließend an die Adresse, die in HL angegeben ist (KLICK)

;Auszugebender Text in diesem Programm;

;-----;

;Der auszugebende Text besteht aus Steuerzeichen, deren Parameter-Bytes
;und aus normalen Textzeichen wie 0-9 und A-Z etc.

ESM DB &1E,16,3 ;Der Text beginnt mit dem Control Code &1E, er positioniert
;den Cursor auf dem Bildschirm. Dabei wird vorausgesetzt, dass
;sich der Bildschirm im 64 x 32 Modus befindet.
;Dem Control Code &1E (LOCATE) folgen zwei Bytes, die Y und X
;Positionen angeben, auf die der Cursor gesetzt werden soll.

DB "Mit <SHIFT> und <CONTROL> wird das Verzeichnis selektiert." ;Text ausgeben

DB &1E,18,6 ;Cursor auf 18. Zeile und 6. Spalte setzen.

DB "Bitte selektieren Sie das zu ladende HinterGrundBild" ;dort Text ausgeben

DB &1E,20,6,"(Extension HGB) im Datei-Fenster des Turbo-Desktop." ;Y=20, X=6

DB &1E,22,13,"Nach der Auswahl des Hintergrundbildes" ; Y Pos. = 22, X = 13

DB &1E,24,15,"klicken Sie bitte auf das OK-Icon."

DB 0 ;Der Text wird durch ein Null-Byte beendet.

```
;Der folgende Text wird als Fehlermeldung ausgegeben,  
;wenn keine Datei markiert wurde.
```

```
NOFILE DB &1E,30,11,"Sie haben kein HintergrundBild selektiert!",0 ;Y=30, X=11
```

```
;Der folgende Text wird als Fehlermeldung ausgegeben,  
;wenn kein freies Erweiterungs-RAM gefunden wurde.
```

```
NOERAM DB &1E,30,7,"Es wurde kein freies E-RAM fuer das HGB gefunden!",0
```

```
;Der folgende Text wird als Fehlermeldung ausgegeben,  
;wenn beim LADEN des HGB ein Fehler aufgetreten ist.
```

```
LADERR DB &1E,30,7,"Beim LADEN des Bildes ist ein Fehler aufgetreten!",0
```

```
;Nachfolgend werden die LABELs der System-Variablen und Funktionen aufgelistet.  
;Diese EQUates sind direkt von der Datei #E.D übernommen.
```

```
;EQUates der Variablen des FutureOS
```

```
RAMCHAR EQU &B847 ;Alle diese Variablen befinden sich im 64 KB Hauptspeicher  
AKT_RAM EQU &B84C  
REG08_3 EQU &B8DF  
REG16_1 EQU &B8E6  
OK_ADR EQU &BA58  
OK_ATE EQU &BA5A  
OK_BLK EQU &BA5C  
OK_BTE EQU &BA5E  
HGB_RB EQU &BB9A  
HGB_ST EQU &BB9B
```

```
;EQUates der Systemfunktionen des FutureOS
```

```
ROM_D EQU &FF12 ;Funktionen in allen ROMs (identischer Bereich in allen ROMs)  
ROM_A2C EQU &FF2A
```

```
TERM_2I EQU &D358 ;Funktionen in ROM A  
TERM_2 EQU &D48C  
EFER EQU &D9C1
```

```
LADEN EQU &FD8F ;Funktionen in ROM C  
FMD32 EQU &FD98
```

```
KLICK EQU &FE9A ;Funktionen in ROM D  
TUR_E EQU &FE9D
```

```
LIST ;Das Auflisten des Quellcodes beim Assemblieren wieder einschalten. Denn...  
DUMP ;Befehl erstellt eine Liste aller LABELs und zeigt sie am Bildschirm an
```

Assemblieren eines Programms direkt ins RAM

Wenn man einen Routine, eine Funktion oder ein kurzes Programm testen will, dann ist es effizienter das Programm direkt ins RAM zu assemblieren. Man verzichtet also darauf das Programm beim assemblieren auf Diskette bzw. Festplatte zu speichern und anschließend von dort zu laden.

Was man bei direkter Assemblierung beachten muss

Wenn man ein Programm (z.B. unter MAXAM) direkt assembliert, dann wird es direkt ins RAM geschrieben. Ein solches Programm erkennt man daran, dass es keinen „**WRITE**“ Befehl im Quell-Code enthält. So ein Programm wird ab der durch den Befehl „**ORG**“ definierten Adresse ins RAM geschrieben.

Worauf sollte man nun achten?

Will man ein Programm direkt assemblieren, so lädt man es von Diskette, assembliert es und ruft FutureOS mit dem Befehl **|OS** bzw. **|FDESK** auf. Unter FutureOS wird dann das Programm aufgerufen. Diese Vorgehensweise wird anschließend im Detail erklärt.

Beim Start des OS mittels des Befehls **|OS** bzw. **|FDESK** werden mehrere Bytes an einigen Adressen im RAM verändert. Deshalb kann man sein Programm nicht DIREKT an JEDE beliebige Adresse assemblieren. Folgende Adressen bzw. Bereiche werden durch den Aufruf des OS verändert:

- **Adresse &0038:** An diese Adresse schreibt das OS das Byte &C9 (dies ist der Opcode für den Assembler Befehl RET). Die maskierbaren Interrupts werden dadurch blockiert.
- **Adressen &0066 & &0067:** Diese beiden Speicherstellen werden mit den Bytes &ED, &45 beschrieben (diese zwei Bytes entsprechen dem Opcode für den Assembler Befehl RETN). Auch die unmaskierten Interrupts werden dadurch blockiert.
- **Der Bereich oberhalb von &A000** wird mit &00 initialisiert, denn ab &A000 befinden sich im FutureOS die File-Tagging-Bytes (FTBs), der Text-Bildschirm, die System-Variablen, Datei-Header und der Stack. Ab &C000 folgt nur noch der Bildschirmspeicher.

Wohin soll ich also direkt assemblieren?

Es empfiehlt sich sein Programm ab Adresse &9000 (mittels des Befehls „**ORG &9000**“) in den Speicher zu assemblieren. In diesem Fall kann das Programm den Bereich von &9000 bis &9FFF belegen, ohne beim Start des OS verändert zu werden. Dieser 4 KB lange Bereich sollte zum Testen von Routinen, Funktionen oder auch kurzer Programme ausreichend sein.

Sollte das Programm über 4 KB anwachsen, dann empfiehlt es sich den Quell-Code auf Diskette zu assemblieren und das generierte Programm anschließend unter FutureOS zu starten. Dabei steht dem Programm der gesamte Speicher von &0000 bis &9FFF zur Verfügung (und zwar inklusive der Interrupt-Einsprünge und RST-Einsprünge). Genaueres dazu findet man im ersten Kapitel.

Wie man ein Programm direkt in das RAM assembliert

1. Bei Verwendung von MAXAM 1.14

- a. Der Quell-Code wurde entweder gerade unter MAXAM eingegeben, oder im Text-Editor-Menü von MAXAM mittels des Kommandos „L“ von Diskette geladen.
- b. Falls der Quell-Code frisch eingegeben wurde, so muss er noch auf Diskette gesichert werden. Dies geschieht im Text-Editor-Menü mittels des Kommandos „S“.
- c. Im Text-Editor-Menü des MAXAM Assemblers geben Sie nun den Befehl „A“ ein, gefolgt von Return. MAXAM beginnt nun zu assemblieren.
- d. Nach dem Ende der Assemblierung drücken Sie bitte eine Taste um ins Text-Editor-Menü von MAXAM zurückzukehren. Das assemblierte Programm steht nun im RAM.
- e. Bitte geben Sie im Menü nun den Befehl „X“ ein, dadurch gelangen Sie in den RSX Befehlsmodus. Der RSX-Befehlsmodus ist daran zu erkennen, dass sich am Anfang jeder Zeile (links vom Cursor) der RSX Strich „|“ als Prompt befindet.
- f. Geben Sie nun bitte „OS“ ein und drücken Sie Return. Sie gelangen ins FutureOS.
- g. Unter FutureOS bewegen Sie bitte den Mauszeiger zum „RUN“ Icon und klicken es an (Es befindet sich ganz links in der untersten Reihe).
- h. Nun tippen Sie auf die Taste „3“ um ein RAM Programm zu starten.
- i. Sie werden nach der Start-Adresse gefragt. Die Start-Adresse ist normalerweise die Adresse ab der Sie assembliert haben (in unserem Beispiel also &9000). Bestätigen Sie mit Return.
- j. Schlussendlich werden Sie auch noch nach dem RAM Block gefragt. Bitte geben Sie den Wert &C0 ein. Dadurch werden beim Aufruf des Programms die ersten 64 KB eingeblendet (Durch die Eingabe anderer Werte lässt sich die RAM Konfiguration vor dem Aufruf des Programms verändern).

2. Bei Verwendung von MAXAM 1½ und Protext

- a. Der Quell-Code wurde entweder gerade unter Protext eingegeben, oder im Befehls-Modus von Protext mittels des Kommandos „L“ von Diskette geladen.
- b. Falls der Quell-Code frisch eingegeben wurde, so muss er noch auf Diskette gesichert werden. Dies geschieht im Befehlsmodus von Protext mittels des Kommandos „S“.
- c. Im Befehls-Modus von Protext geben Sie nun den Befehl „asm“ ein, gefolgt von Return. Dadurch wird der MAXAM Assembler aufgerufen und beginnt nun zu assemblieren. Das assemblierte Programm steht nun im RAM.
- d. Bitte folgen Sie nun den Schritten ab Punkt „f“ in der obigen Beschreibung.

Anmerkung: Der Quell-Code des assemblierten Programms darf beim direkten Assemblieren keine „WRITE“ Anweisung enthalten! Denn sonst würde der Code auf Diskette geschrieben werden, anstatt im RAM abgelegt zu werden.

Das RSX Kommando |FDESK: In dem hier gezeigten Beispiel wird FutureOS mit „|OS“ aufgerufen. Es ist allerdings auf möglich das OS mittels des Befehls „|FDESK“ aufzurufen. Dabei wird der Hauptspeicher des CPC erhalten, und nach dem Verlassen von FutureOS mittels des „END“ Icons steht der Quellcode sofort wieder zur Verfügung. Aber Achtung, es kann hier manchmal zu Problemen kommen. In jedem Fall sollte man seinen Quell-Code immer auf Disk speichern.

Direkt assemblieren für Fortgeschrittene

Im vorigen Kapitel wurde bereits gezeigt, welche Adressen bzw. Speicher-Bereiche beim Start von FutureOS überschrieben werden. Alle anderen Bytes zwischen &0000 und &9FFF stehen dem Programmierer prinzipiell für seine Programme zur freien Verfügung. Auch das freie E-RAM kann verwendet werden. Jedoch gibt es hier einige Dinge zu beachten:

- Da das OS den unteren 16 KB Block als Sortier-Puffer für Dateinamen benutzt, ist es nicht ratsam sein Programm direkt dorthin zu assemblieren. Bei jedem Einlesen eines Inhaltsverzeichnisses kann dieser Speicherbereich überschrieben werden. Normalerweise ist dies nicht der Fall, es kann jedoch bei Speichermangel dazu kommen.
- Ein weiterer Grund, warum man den unteren 16 KB Block (&0000 bis &3FFF) nicht benutzen sollte, stellt MAXAM selbst dar. Denn MAXAM legt ab ca. Adresse &0172 auch den Quell-Code des Programms ab. Beim Assemblieren in den unteren RAM Block würde man also seinen eigenen Quellcode überschreiben, und das würde vermutlich zum Absturz des CPC führen.
- Im Bereich von &4000 bis &7FFF findet das Banking des E-RAMs statt! Falls man also mit E-RAM arbeitet, so sollte man sein Programm nicht in diesem Block ablegen. Jedenfalls nicht komplett.
- Achtung: Beim Laden oder Speichern von Dateien unter FutureOS dient der Bereich von &8000 bis &8FFF manchmal als Sektor-Puffer.

Will man also auf Nummer sicher gehen, dann sollte man ab &9000 assemblieren. Es stehen immerhin 4 KB zur Verfügung. Dieser Bereich von &9000 bis &9FFF wird vom OS nicht angetastet. So ist es beispielsweise möglich ein Programm zu assemblieren, anschließend das OS aufzurufen, dann kann man z.B. diverse Daten laden und schlussendlich das Programm an Adresse &9000 aufzurufen.

Der Bereich von &9000 bis &9FFF ist zwar "nur" 4 KB groß, aber für 4 KB Code muss man schon mit großen Quell-Code-Dateien arbeiten! Die Variablen des Programms können an beliebiger Stelle abgelegt werden, sie müssen nicht Teil des Programms sein. Entsprechende Variablen müssen natürlich vom Programm initialisiert werden.

Wenn man auf Interrupts verzichten kann und diese deaktiviert lässt, und weiterhin keine RST Befehle benötigt, so kann man seine Variablen und/oder Puffer ab Adresse &0000 anlegen. Genauso ist es problemlos möglich den ersten RAM Block (&0000 bis &3FFF) als zweiten Bildschirm zu nutzen.

Es steht dem Programm frei, das E-RAM zu nutzen, man sollte allerdings auf die Memory Management Funktionen des OS zurückgreifen, um sich einzelne 16 KB RAM Blöcke zuteilen zu lassen. Welche 16 KB Blöcke des Erweiterungs-RAMs belegt sind kann man durch Auslesen und Auswerten der Variablen XRAM_C4 bis XRAM_FF ermitteln.

Temporär kann auch der Bereich von &A000 bis &AFFF verwendet werden. In diesem Fall stehen dem Programm 44 KB Hauptspeicher am Stück zur Verfügung. Sobald allerdings Disketten-Betrieb stattfindet können einzelne Bytes oberhalb von &A000 verändert werden. Ab &A000 befinden sich die FTBs, darin enthaltene Bytes werden durch Änderung des Status einer Datei verändert.

Weiterhin lässt sich auch der Bereich von &B000 bis &B7FF temporär als Datenspeicher nutzen, dieser Bereich wird allerdings von einigen Floppy-Disc-Routinen überschrieben.

Wir programmieren unser erstes Spiel

Pong!

Lustige Idee: Beim Pong wird nach oben gespielt, und wie bei Tetris werden die ganzen Icons weggekickt.

((Lustige Zusatzidee, am Ende (keine Icons mehr da) wird das SOS Desktop eingescrollt, und u. U. sogar mit nativer Funktionalität ausgestattet.))

((Oder: Meldung: Your OS gets an upgrade, SOS screen scrollt rein. Dann Fehlermeldung, sorry CPC was infected by a virus, deleting virus... Upgrade comes now: Ganz viel Icons werden dargestellt, oder sonst was hübsches ;-))

Sind alle Icons weg, dann werden die Icons von neuem (anders verteilt?) von oben eingescrollt.

Der Ball wird Mode 2 Pixel-genau dargestellt. Möglich das sich der Ball innerhalb von 3 x 2 Mode 2 Zeichen bewegt. Der Ball kann also mit veränderten Textzeichen dargestellt werden. Ebenso wie der Schläger ;-)