4. Betriebssystemroutinen

Um als Programmierer das Rad nicht ständig neu erfinden zu müssen, stellt uns das Betriebssystem *Programme* zur Verfügung, welche wir von unserem Programm aus anspringen können. Diese werden auch *Unterprogramme* oder *Betriebssystemroutinen* genannt. Das kann uns eine Menge Arbeit ersparen. Die ARM hat dafür einen eigenen Befehl: `&EF`, gefolgt von der Nummer der Routine oder des Unterprogramms. Der Befehl muß in dieser hexdezimalen Schreibweise jedoch wie immer *achtstellig* sein!

In Assembler schreibt man dafür auch `SWI`. `SWI` steht für *Software Interrupt*. Wir werden bei dieser speziellen Art von Unterprogrammen daher auch von SWIs reden.

Bei diesem Befehl, `&EF`, schaut der Prozessor an Hand der folgenden Nummer in einer langen Liste nach, wo er im Speicher das Unterprogramm findet. Dieses springt er dann an und arbeitet es ab. Ist das Unterprogramm beendet, setzt er das ursprüngliche Programm wieder fort. Dazu wird, wie bereits erwähnt, der Wert im Programmzähler verändert.

Solche Betriebssystemroutinen oder Unterprogramme bzw. SWIs können ohne oder mit Paramter aufgerufen, d. h. angesprungen werden. Für RISC OS werden solche Betriebssystemroutinen unter der Bezeichnung SWI im Programmer's Reference Manual beschrieben.

4.1 Betriebssystemroutinen ohne Parameter

Wir schauen uns der Einfachheit halber zuerst zwei SWIs an, welche ohne Parameter auskommen. Sie haben die hexdezimalen Nummern `&406C0` und `&406Cl`. `&406C0` schaltet die Sanduhr ein, `&406Cl` schaltet sie wieder aus. Probieren wir das also einmal in der Praxis aus und geben zwei kleine Maschinenprogramme in !StrongEd oder in !Zap wie unter Abschnitt 3.1 beschrieben ein:

```
EF04 06C0
ElA0 F00E
```

`Listing 4.1.1`

```
EF04 06Cl
ElA0 F00E
```

`Listing 4.1.2`

Nicht vergessen: Die Befehle müssen in der hexdezimalen Darstellung immer acht Zeichen lang sein. Der eigentliche Befehl `EF` steht ganz links. Die fehlenden Stellen zwischen dem Befehl und der Nummer des SWIs müssen wir deshalb immer mit Nullen auffüllen! Die kleinste Nummer fängt nämlich rechts an, nicht links. Die Zahlen bauen sich dann wie gewohnt von rechts nach links auf, d. h. der Übertrag wird immer links an der vorherigen Stelle angefügt.

Diese Programme müssen wir wieder in zwei Dateien speichern, bevor wir sie jeweils mittels einem Doppelklick starten können. Siehe hierzu auch den Abschnitt 4.1.

Starten wir Listing 4.1.1 so verwandelt sich der Mauszeiger in eine Sanduhr. Starten wir Listing 4.1.2, so verwandelt sich die Sanduhr wieder in den Mauszeiger zurück.

Dies sind die ersten zwei lauffähigen Programme, welche irgendwas bewirken, was der Anwender auch sieht. Beide Programme sind jeweils 8 Bytes groß. So einfach ist das!

In Assembler können wir ebenfalls die Nummer des SWIs verwenden. Allerdings soll man laut diversen Lehrbüchern nicht die Nummer, sondern den Namen des SWIs angeben. Dieser Name wird vom Assembler dann in die Nummer des SWIs umgerechnet und ist im Programmer's Reference Manual festgelegt. Der Prozessor selbst kann mit dem Text nichts anfangen. Er braucht die Nummer.

Listing 4.1.2 sähe im Assembler des BBC-BASICs so aus:

```
DIM code% (100)
FOR pass = 0 TO 3 STEP 3
P% = code%
[
 OPT pass
 .start
 SWI "Hourglass_On"
 MOV PC, R14
 ]
NEXT pass
PRINT "Startadresse &:" ~code%
PRINT "Programmgröße ist &"; P%-start; "Bytes lang"
END
```

Listing 4.1.3

Listing 4.1.3 macht nur Sinn, wenn auch der Mauszeiger zu sehen ist. Falls der Mauszeiger abgeschaltet ist, z. B. weil man durch Druck auf die Funktionstaste F12 von der WIMP auf die Befehlszeile gewechselt hat, kann man mittels dem Befehl `*pointer 1` zuvor den Mauszeiger aktivieren.

Man kann in Listing 4.1.3 nun noch die Nummern `&406C0` und `&406C1` statt den Texten "Hourglass_On" und "Hourglass_Off" ausprobieren und auch versuchen, die Programme 4.1.1 bzw. 4.1.2 mit dem Acorn Assembler oder der GCC zu erstellen. Als Grundlage hierfür mögen die Assemblerbefehle aus Listing 4.1.3 hilfreich sein. Das &-Zeichen ist dabei stets mit einzugeben.

Beim Acorn Assembler (DDE) müssen wir berücksichtigen, daß wir dem Assembler erst die

Namen der SWIs bekannt machen müssen (falls wir diese verwenden wollen). Diese Namen sind in der Datei

```
DDE.Sources.DDE-Examples.ObjAsm.AsmHdrs.h.SWINames
```

zu finden. Am besten, man kopiert diese Datei mit den übergeordneten Verzeichnissen `AsmHdrs.h.SWINames` ins aktuelle Verzeichnis und bindet diese Datei mit dem Befehl `GET AsmHdrs.h.SWINames ins` Quellprogramm mit ein. Leider sind diese Art von Listen nicht im Programmverzeichnis vom Acorn Assembler enthalten. Diese unschöne Art wurde wohl von Unix übernommen.

Beim Befehl `GET` handelt es sich *nicht* um ein Mnemonic! Also um keinen Befehl, welcher der Prozessor in irgend einer Art und Weise kann. Dieser Befehl wird daher auch nicht in Maschinensprache übersetzt. Er macht nur dem Assembler etwas bekannt, damit er an Stelle des Namens die entsprechende Nummer verwenden kann. Sonst kann er mit dem Namen nichts anfangen.

Außerdem ist zu berücksichtigen, daß der Name des SWIs im Acorn Assembler im Vergleich zum BASIC Assmbler *ohne* Anführungszeichen eingegeben werden muß! Groß- und Kleinschreibung sind ebenfalls zu beachten!

```
      AREA     |main|, CODE

      GET AsmHdrs.h.SWINames

      ENTRY
      SWI Hourglass_On
      MOV pc, rl4
      END
```

Listing 4.1.5

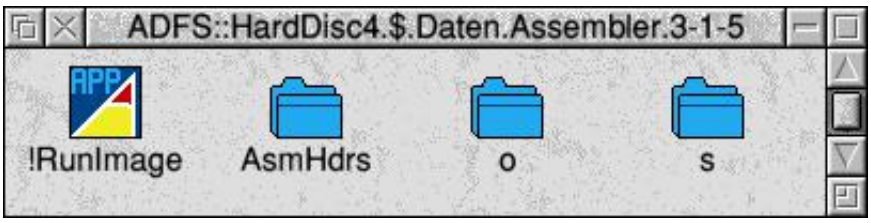

Bild 4.1: Das Arbeitsverzeichnis von Listing 4.1.1.5 für den Acorn Assembler mit dem bereits in Maschinensprache übersetzten Programm

Bei Verwendung der GCC müssen die SWIs noch einmal anders behandelt werden. Am einfachsten ist es, man schreibt einfach die Nummer des SWIs in hexdezimaler Form hinter den Befehl. Allerdings muß diese Nummer hier statt mit einem & mit einem 0x beginnen.

```
   .global _start

_start:
    SWI 0x406C0
    MOV PC, R14
```

Listing 4.1.6

Übersetzt (compiliert) wird dieser Quellcode wie unter Abschnitt 3.4 für die GCC beschrieben.

Will man bei der GCC die Namen der SWIs verwenden - ich habe dazu nichts gefunden. Der Befehl `GET` und die Datei `AsmHdrs.h.SWINames` für den Acorn Assembler aus Listing 4.1.5 scheint die GCC jedenfalls nicht zu verstehen.

4.1.2 SWIs mit einem einzigen Parameter

Parameter werden üblicherweise mit den Registern des Prozessors übergeben. Das heißt, man füllt zuerst die Register des Prozessors mit den entsprechenden Daten. Anschließend ruft man den entsprechenden SWI auf.

Der erste SWI mit der Nummer `&0` gibt - wen sollte es wundern! - ein Zeichen auf den Bildschirm aus. Klar, man möchte ja etwas auf dem Bildschirm sehen! Deshalb ist dieser SWI so wichtig. Das Zeichen, welches man ausgeben möchte, muß ASCII-codiert im Register R0 stehen.

Das kleine 'a' hat den ASCII-Wert 65 oder hexdezimal 41. Das geht dann so:

Maschinensprache:	*Assembler:*
`E3A0 0041`	`MOV R0, #&41`
`EF00 0000`	`SWI OS_WriteC`
`E1A0 F00E`	`MOV PC, R14`
`Listing 4.2.1 und`	`Listing 4.2.2`

Wie man beim Vergleich von Listing 4.2.1 und Listing 4.2.2 erkennen kann, gibt es in Maschinensprache zwei verschiedene MOV-Befehle. Der eine Befehl (E1) überträgt den Wert von einem Register in ein anderes. Der andere Befehl (E3) aber schreibt einen festen Wert in ein Register. Dieser Wert ist Bestandteil von dem Befehl, der im Speicher steht und *vier* Bytes umfaßt.

Nun gibt es hierbei das Problem, daß dieser Wert nur ein Byte (8 Bit) umfassen kann. Denn die anderen drei Bytes werden für den Befehl selbst gebraucht. Ein Register ist aber 32 Bit

breit! Wir können mit diesem Befehl (E3) allein also kein Register füllen.

Im Falle vom `SWI 0` oder `OS_WriteC` ist das nicht schlimm, weil diese Routine nur den (erweiterten) ASCII-Satz verarbeiten kann und dieser (erweiterte) ASCII-Satz nur 1 Byte (oder acht Bit) umfaßt.