Die Ebenen des Absturzes

Nachdem ich mir den Commodore 64 gekauft hatte, schnappte ich mir das Handbuch und fing an, in Basic zu programmieren. Wenn ich einmal versehentlich in eine Endlosschleife geriet, drückte ich die Run/Stop-Taste, und das Programm wurde unterbrochen, damit ich den Fehler beseitigen konnte — die Computerwelt war in Ordung!

Doch wer den C 64 kennt, hat sich sicher schon an die Befehle PEEK und POKE versucht. Zum Beispiel:

```
100 FOR I = 0 TO 999
110 POKE 1024 + I,0
120 POKE 55296+I,0
130 NEXT I
```

Dieses kleine Programm füllt den Bildschirm mit schwarzen Klammeraffen. Doch wehe, man hat in Zeile 100 statt 999 die leicht erweiterte Version 9999 stehen! Direkt hinter dem Bildschirmspeicher liegt der Basic-Benutzerspeicher. Bei solchen Fehlern frißt sich das Programm von selbst auf. Das Programm ist teilweise, wenn nicht ganz, zerstört. Der fortgeschrittene Programmierer macht sich sicher eines Tages Gedanken, wie er seine Programme durch versehentliches Drücken der Run/Stop-Taste schützen kann.

Dieses ist besonders dann angebracht, wenn das fertige Programm von einem C 64-Unkundigen bedient werden soll.

Dieses kleine Porgramm bewirkt Wunder:

```
100 FOR I=830 TO 834
110 READ A : POKE I,A : NEXT I
120 POKE 808,62 : POKE 809,3
130 DATA 169,1,201,0,96
```

Weshalb? Im Betriebssystem gibt es ein Unterprogramm, welches die Stopp-Taste abfragt. Das Programm liegt zwischen 63213 ($ F6ED) und 63226 ($ F6FA). Die Anfangsadresse dieser Routine ist in den Speicherzellen 808 und 809 gespeichert; und zwar Low-Byte vor High-Byte. Nun wird diese Routine regelmäßig aufgerufen. Ist die Stopp-Taste gedrückt worden, so veranlaßt diese Routine — abgesehen von ein paar anderen Instruktionen —, daß das Zeroflag gesetzt wird. Wurde die Stopp-Taste aber nicht gedrückt, so löscht dieses Unterprogramm das Zeroflag. Wenn die Speicherzellen 808 und 809 so verändert werden, daß zu einer Routine gesprungen wird, die immer das Zeroflag löscht, dann wird nie das Drücken der Stopp-Taste erkannt. Dieses geschieht durch unser kleines Programm, welches eine Routine in den Kassettenpuffer schreibt.

Haben wir für kritische Programmteile die Stopp-Taste unterdrückt, können wir jene wieder durch den Befehl

```
200 POKE 808,237 : POKE 809,246
```

aktivieren.

Nehmen wir einmal an, uns ist folgendes passiert: Wir haben ein Programm geschrieben, das die Stopp-Taste ausschaltet (damit auch Run/Stop + Restore). Das Programm stürzt ab (zum Beispiel durch eine Endlosschleife), und wir haben es noch nicht abgespeichert. Das Programm scheint verloren, denn wir können den Computer nur noch aus- und einschalten. Doch da gibt es noch eine Rettung. Das magische Wort heißt RESET.

Falls Sie in einem Programm einen Reset wünschen, dann benutzen Sie folgenden Befehl:

```
100 SYS 64738
```

Der Bildschirm verengt sich links und rechts um ein halbes Zeichen, und dann meldet sich der Computer, als ob sie ihn gerade eingeschaltet hätten. Hardwaremäßig bewirken sie einen Reset, indem Sie Pin 1 und Pin 3 am User-Port verbinden.

Sie können aber auch Pin 2 und Pin 6 am seriellen Bus verbinden. Dazu nehmen Sie das Kabel aus dem Floppy-Laufwerk und verbinden die beiden Pins. Das Prinzip ist, die Reset-Leitung mit der GND-Leitung (Erde) zu verbinden.

Komfortabler geht es mit einem Resetschalter, der in den User-Port eingeschoben wird und per Knopfdruck einen Reset bewirkt.

Wenn Sie nun versuchen, Ihr Programm zu listen, wird Ihre Skepsis zunächst bestätigt werden. Es gibt scheinbar kein Programm mehr.

Durch das Rücksetzen (Reset) des Computers sind alle Basic-Pointer auf ihren Urzustand gebracht worden. Das Programm selbst existiert noch, denn der Basic-Benutzerspeicher wurde nicht gelöscht. Um die Zeiger (Pointer) zurückzusetzen, und somit Ihr Programm wieder sichtbar zu machen, laden Sie das Programm »UNNEW« ein, das Sie hoffentlich vorher eingetippt haben. Nun starten Sie es und — das Basic-Programm wird wieder sichtbar.

Haben Sie vor dem Reset in Maschinensprache mit einem Monitor (zum Beispiel HES-Mon oder Supermon 64) programmiert, dann starten Sie den Monitor und fahren Sie mit der Programmierung beziehungsweise dem Testen fort. Sowohl Ihr Maschinenprogramm als auch der Monitor wurden nicht zerstört.

Haben Sie sich einmal mit dem Umgang des Resets vertraut gemacht, werden Sie sicher auch gerne einmal hinter die Kulissen von Videospielen schauen wollen. Dieses konnten Sie

bisher nicht, weil viele Programme die Run/Stop-Taste sperren. Funktioniert der Reset, dann können Sie mit einem Monitor das Spiel erforschen. Doch was geschieht, wenn der Reset wirkungslos bleibt? Wie kann sich ein Videospiel gegen das scheinbare Aus- und Einschalten schützen?

Die Lösung besteht in der Möglichkeit, Module anzuschließen, nämlich wenn Sie ein Modul einschieben und das Gerät einschalten, dann meldet sich keineswegs Basic V2.0 mit einem READY, sondern das Modul übernimmt das Kommando, ohne daß Sie es dazu aufgefordert hätten.

Dazu muß das Modul drei Sachen veranlassen:
1. Den Basic-Interpreter ausblenden,
2. sich als Modul zu erkennen geben und
3. eine Einsprungadresse zur Verfügung stellen.

Das Wegbleiben eines ROMs, sei es der Basic-Interpreter oder das Betriebssystem, erfolgt durch Setzen beziehungsweise Löschen von Bits in der Speicherzelle 0001. Dort liegen die für uns interessanten Kanäle High-RAM und Low-RAM, die durch die beiden Bits 0 und 1 dargestellt werden. Im Einschaltzustand sind beide gesetzt. Ferner gibt es die hardwaremäßigen Kanäle GAME und ExROM. Diese liegen im Moduleinschub und sind ohne Modul auf High (1) gesetzt. Wird ein Modul eingeschoben, so bewirkt dieses, daß die Kanäle GAME und ExROM auf Low gesetzt werden. Dadurch wird das Basic-ROM ausgeblendet.

Als zweites muß sich das Modul zu erkennen geben. Im Betriebssystem gibt es eine Routine, die erkennt, ob ein Modul eingeschoben ist. Sie liegt zwischen 64770 ($ FD02) und 64788 ($ FD14). Dabei wird überprüft, ob in den Speicherzellen 32772 bis 32777 ($ 8004 bis $ 8009) das Wort »cbm80« steht. Das sind die ASCII-Zeichen: 195,194,205, 56,48. Ist das der Fall, setzt das Unterprogramm das Zeroflag. Wenn nun eine Routine im Betriebssystem (zum Beispiel die Reset-Routine) dieses Unterprogramm aufruft, und das Zeroflag wird gesetzt, dann nimmt das Betriebssystem an, daß ein Modul vorliegt. Es springt zur Adresse, die in $8002 und $ 8003 steht. Die Steuerung wird dem Modul übergeben. Will man nun ein Modul simulieren, muß man
1. $ 8004 bis 8009 mit »cbm80« belegen,
2. eine Adresse in $ 8002 und $ 8003 eintragen (Low-Byte vor High-Byte) und
3. durch den Befehl CLI Interrupts (IRQ) erlauben.

Mit folgendem kleinen Programm sorgt man von Basic aus dafür, daß ein Reset keinen Effekt hat:

```
100 FOR I=32770 TO 32778
110 READ A : POKE I,A
120 NEXT I
130 DATA 10, 128, 195, 194, 205
140 DATA 56, 48, 88, 0
```

Unser Ausgangsproblem war aber: Wie kann ich ein Reset verursachen, obwohl es gesperrt ist? Es gibt zwei Methoden:
1. Man sorgt dafür, daß in $ 8004 bis $ 8009 nicht »cbm80« steht.
2. Die Routine im Betriebssystem ändert man so ab, daß sie nicht mehr »cbm80« sondern zum Beispiel »cbm81« abfragt.

Es soll zuerst der zweite Fall behandelt werden. Da das Betriebssystem (auch Kernal genannt) im ROM liegt, läßt es sich nicht ohne weiteres ändern. Deshalb wird es zuerst in das darunterliegende RAM kopiert. Dieses funktioniert folgendermaßen:

```
100 FOR I=57344 TO 65535
110 POKE I,PEEK(I)
120 NEXT I
```

Nun könnte, durch Löschen des Bits 1 (High-RAM) in der Speicherzelle 0001, vom ROM auf RAM umgeschaltet werden:

```
320 POKE I,PEEK(I) AND 253
```

Durch das Löschen dieses Bits wird aber auch das Basic-ROM ausgeblendet. Also müssen wir dieses auch ins RAM kopieren:

```
200 FOR I=40960 TO 49151
210 POKE I,PEEK(I)
220 NEXT I
```

Jetzt könnte man das Programm starten, es hätte jedoch keinen Effekt, weil in der Adresse 0001 immer die Zahl 55 (High-RAM gesetzt) erscheint — statt der angestrebten 53. Dieses kommt durch den I/O-Reset. Diese Routine liegt im Kernal und setzt die Kanäle in 0001 immer wieder neu. Also muß man diese Routine für unsere Zwecke ändern:

```
300 POKE 64982,229
```

Schließlich werden wir aus der »cbm80«-Abfrage eine »cbm81«-Abfrage machen:

```
310 POKE 64788,49
```

Wenn man jetzt das Programm startet und ein Reset auslöst durch: SYS 64738 wird sich die Maschine zurücksetzen, auch wenn in 32772 bis 32777 (exklusive) »cbm80« stehen sollte. Betätigt man jedoch den Resetschalter, dann wird der Reset abhängig von 32772 bis 32777 ausgelöst. Beim Reset werden nicht nur der Prozessor, sondern auch die beiden CIAs und der SID zurückgesetzt. Diese bewirken, daß in 0001 High-RAM gesetzt wird, und somit kommt die Veränderung des Betriebssystems nicht zum Tragen. Im ersten Fall wird ein POKE 32772,0 das Nötige tun, sofern nicht das Programm seinerseits dafür sorgt, daß »cbm80« immer wieder erneuert wird.

Hier das Programm UNNEW:

```
100 FOR I=525 TO 578
120 READ A : POKE I,A : NEXT I
200 POKE 43,525 AND 255 : POKE 44,2
210 POKE 45,578 AND 255 : POKE 46,2
220 CLR : SAVE "UNNEW" ,8 : REM bzw. ,1,1
300 DATA 160,003,200,177,043,208,251,200
310 DATA 200,152,160,000,145,043,165,044
320 DATA 200,145,043,133,060,160,000,132
330 DATA 059,162,000,200,208,002,230,060
340 DATA 177,059,208,245,232,224,003,208
350 DATA 242,200,208,002,230,060,132,045
360 DATA 164,060,132,046,096,256
```

Wenn Sie dieses Programm eingeben und starten, wird es ein Programm namens »UNNEW« auf Diskette schreiben. Falls sie aus Versehen NEW eingetippt haben, dann laden sie das Programm durch

```
LOAD "UNNEW" ,8,1
```

und starten es durch

```
SYS 525
```

Ihr Basic-Programm ist gerettet, selbst wenn sie ein Reset ausgelöst haben. Das Programm setzt die Zeiger in $ 0801 und $ 0802, die auf die nächste Zeile zeigen, auf den richtigen Wert. Bei NEW werden diese beiden Bytes auf Null gesetzt, und zwei aufeinanderfolgende Nullen bedeuten für den Interpreter »Ende des Programms«.

(Daniel Kossmann/aa)