

Für viele Leute ist sie die letzte Rettung in der Not, für den Programmierer hält sie eine Menge an unliebsamen Überraschungen bereit. Sie ahnen es sicher schon, es geht um die RESET-Taste.

### Die RESET-Taste

---

Normalerweise kümmert man sich um diese Tatsache kaum, denn sie ist ja nur als Notbremse gedacht. Etwa, wenn sich ein `USR`-Befehl verirrt oder wenn die gerade geschriebene Maschinenroutine mit Basic ganz und gar unverträglich ist. Es gibt aber dennoch mindestens zwei Fälle, in denen RESET einiges Kopfzerbrechen beschert. Zum Beispiel dann, wenn man ein Programm geschrieben hat, das kommerziell verwertet werden soll. Ein kommerzielles Programm unterscheidet sich von seinem Hobby-Verwandten gerade durch die vielen Kleinigkeiten, die es gegen Bedienungsfehler schützen. Und eine der wichtigsten Kleinigkeiten ist der Schutz gegen RESET.

Der zweite Fall ist weitaus häufiger, denn er betrifft Erweiterungen im DOS und im Betriebssystem. Solche Erweiterungen (man denke z.B. an die RAMdisk oder Basic-Befehlserweiterungen) sollten stets so geschützt sein, daß sie vom Anwenderprogramm nicht überschrieben werden können. Man erreicht das beim Atari, indem man die Zeiger auf die oberste oder unterste nutzbare Speicherzelle verändert und somit dem Anwenderprogramm mitteilt, daß weniger Speicher vorhanden ist.

Das ist genau der Punkt, an dem die RESET-Taste ins Spiel kommt. Sobald RESET gedrückt wird, stellt das Betriebssystem und das DOS die Zeiger auf die ursprünglichen Werte zurück. Das Ergebnis ist desolat: Die zuvor geschützten Maschinenroutinen sind quasi zum Überschreiben freigegeben und werden kurz über lang zum Absturz des Rechners führen. Auf der anderen Seite gibt es aber aus gutem Grund keine Möglichkeit, den RESET durch einen einfachen `POKE`-Befehl unwirksam zu machen.

Hier hilft es nur, ein Programm in die für den RESET verantwortliche Routine einzuklinken, das die Zeiger nach einem RESET auf die gewünschten Werte verändert. Aber bevor wir dazu kommen, wollen wir uns erst um den grundsätzlichen Ablauf eines RESET kümmern. Der Druck auf die RESET-Taste des Ataris entspricht in groben Zügen dem, was beim Einschalten des Computers vor sich geht. Die CPU unterbricht sofort das laufende Programm und holt sich eine neue Adresse aus den Speicherzellen `$FFFC`, `$FFFD` ab, mit der sie die Bearbeitung eines Programmes beginnt. Bei den XL/XE-Computern wird durch das RESET-Signal auch in jedem Fall das OS-ROM eingeschaltet.

### Damals und heute

---

Am Rande bemerkt: Die Auslösung des RESETs ist ein Punkt, an dem sich die älteren 400/800-Computer von den heutigen XL/XE Maschinen unterscheiden. Beim 400/800 wurde durch System-RESET ein nicht maskierbarer Interrupt (NMI) ausgelöst. Es hat sich aber herausgestellt, daß der 6502 so böse abstürzen kann, daß er nicht einmal mehr einen NMI annimmt. Daher ist es bei den alten Atari-Computern auch gelegentlich vorgekommen, daß auf System-RESET keine Reaktion erfolgte, etwa wenn die CPU bei einem fehlerhaften Maschinenprogramm auf einen bestimmten ungültigen Code gestoßen ist. Man hat deshalb beim XL/XE den Schaltkreis für RESET geändert. Nun wird anstatt eines Interrupts der tatsächliche RESET-Eingang der 6502-CPU benutzt. Das ist der gleiche Eingang, mit dem der Prozessor nach dem Einschalten in Gang gebracht wird. RESET zeigt daher bei XL/XE in jeder Situation eine Wirkung, wenn es auch nicht immer die gewünschte ist.

---

Der RESET-Vektor in \$FFFC, \$FFFD zeigt auf die POWER-UP-Routine, die sowohl den Kaltstart (Einschalten) als auch den Warmstart (RESET) bearbeitet. Die Unterscheidung wird durch ein Flag vorgenommen: WARMST (\$08), das bei einem Kaltstart Null und einem Druck auf RESET 255 enthält. Dieses Flag wird vom Prozessor am Anfang eines RESET eingerichtet.

Wie aber kann die CPU eigentlich feststellen, ob ein Warm- oder Kaltstart ausgelöst wurde? Das ist gar nicht so einfach, denn beide Möglichkeiten werden auf die gleiche Art und Weise ausgelöst.

Hier wurde ein simpler Trick verwendet: Bei einem RESET (ganz gleich woher) werden die drei Speicherzellen \$33D bis \$33F mit Konstanten aus dem ROM verglichen. Wurde der Computer gerade eingeschaltet, so werden sich dort relativ zufällige Werte befinden. Eine Übereinstimmung mit den drei Konstanten ist ziemlich unwahrscheinlich. Damit lautet die Diagnose auf Kaltstart, und das Flag WARMST wird auf Null gesetzt. Außerdem werden die drei Konstanten ins RAM eingetragen. Ein weiterer RESET findet diese Werte später vor und löst folglich einen Warmstart aus (WARMST = 255). Man nutzt also die Tatsache aus, daß beim Aus- und Einschalten des Computers der Speicher verändert wird, bei Druck auf RESET dagegen nicht.

### RESET Ablauf

---

Kalt- und Warmstart benutzen dieselbe Routine im Betriebssystem, es werden nur einige Teile übersprungen, wenn WARMST ungleich Null ist. Ein Kaltstart löscht z.B. den gesamten Speicher und versucht von Kassette oder Disk zu booten. Beim Warmstart werden dagegen nur Speicherbereiche gelöscht, die vom Betriebssystem (OS) genutzt werden. Außerdem darf natürlich nicht gebootet werden. Schließlich soll dabei das Programm im Speicher erhalten bleiben.

Bei einem Warmstart werden die für das OS reservierten Bereiche (\$10 bis \$7F, Page 2 und Teile von Page 3) gelöscht, die Zeiger auf die unterste und oberste nutzbare Speichergrenze neu gesetzt (!), die Vektorentabelle in Page 2 neu geschrieben, die Tabelle der Gerätetreiber HATABS neu angelegt und ein eventuelles Steckmodul wird neu initialisiert. Anschließend wird der Editor geöffnet, d.h. es wird ein GRAPHICS 0 Bildschirm eingerichtet.

Nun werden die Routinen aufgerufen, die bei Kaltstart einen Kassetten-Boot (nur bei gedrückter Start-Taste) oder einen Disk-Boot ausführen. Finden diese Routinen hingegen das Flag WARMST gesetzt, wird das Booten unterdrückt und nur das evtl. schon geladene Programm initialisiert. Ist dabei das Flag BOOT (\$09) gleich eins, wird angenommen, daß ein Disk-Boot erfolgreich war, und die gebootete Software (im Regelfall das DOS) wird durch einen indirekten Sprung über DOSINI (\$0C, \$0D) initialisiert.

Hat BOOT dagegen den Wert zwei, so wird das von Kassette gebootete Programm über CASINI (\$02, \$03) initialisiert. Ist ein Modul eingesteckt, wird jetzt anhand seines Flagbytes geprüft, ob es gestartet werden will. Ist jedoch kein Modul da, oder will es nicht gestartet werden, so bleibt als letzte Rettung nur der Sprung durch den Vektor DOSVEC (\$0A, \$0B). War ein Disk- oder Kassetten-Boot erfolgreich, so findet sich in DOSVEC die Startadresse der gebooteten Software. Das ist übrigens der Grund, warum Sie beim Booten mit gedrückter OPTION-Taste in's DOS kommen. Wurde nicht gebootet, und ist auch kein Steckmodul vorhanden, so bleibt nur noch der Sprung zum Selbsttest bzw. bei den alten 400/800 zum "Memo-Pad".

---

## RESET-Schutz

---

An der Beschreibung haben Sie sicherlich bemerkt, daß es durch die drei RAM-Vektoren einige Möglichkeiten zum Eingriff in das Geschehen eines Warmstarts gibt. Zuvor aber noch eine Bemerkung zu einem Sachverhalt, der Ihnen im Betriebssystem der Atari-Computer häufig begegnet wird: Der Aufruf eines Programms wird immer in zwei Stufen ausgeführt. Zuerst kommt ein Sprung durch einen Initialisierungs-Vektor, der eigentliche Aufruf erfolgt erst später mit einem Sprung durch einen zweiten Vektor. Wozu diese Komplizierung? Es kommt oft vor, daß mehrere Programme im Speicher des Ataris gleichzeitig vorhanden sind. Denken Sie dabei nur an DOS und Basic. Durch die frühzeitige Initialisierung erhält jedes der Programme Gelegenheit, seine Arbeitsbedingungen vorzubereiten. Im Rahmen der DOS-Initialisierung wird z.B. der DOS-Handler D: in HATABS eingetragen (er wurde beim RESET gelöscht) und die untere Speichergrenze MEMLO auf das Ende von DOS gesetzt. Somit kann es keinen Konflikt mit Basic mehr geben.

Für einen Eingriff ins RESET-Gefüge stehen uns also die Vektoren CASINI, DOSINI und DOS-VEC zur Verfügung. Als erstes und einfachstes Beispiel für ein RESET-sicheres Maschinenprogramm soll der CASINI-Vektor benutzt werden. Sehen Sie sich dazu Listing 1 an. Geben Sie es mit einem Assembler ein (verwendet wurde ATMAS, mit ein paar Änderungen gehen andere auch) und starten Sie es an der Adresse \$600. Als Demo wird hier ein Programm aufgerufen, das eine einfache Display-List aktiviert und ein paar Farben in den Hintergrund zaubert. Nun kommt's: Drückt man RESET, läuft das Programm wieder an. Das bedeutet: Wenn es einmal gestartet wurde, kann man nicht einmal mehr in den Assembler zurückkehren (daher vorher aufzeichnen!).

Folgende Zutaten sind dazu nötig: Das Flag BOOT erhält den Wert 2 und CASINI wird auf das Programm gerichtet, das nach RESET laufen soll. Damit wird der Warmstart-Routine vorgegaukelt, daß ein erfolgreicher Kassettenboot erfolgte, und so springt diese indirekt durch CASINI und startet das Programm.

Eine Konsequenz dieser Methode darf man allerdings nicht vergessen: Das DOS wird dabei nicht initialisiert und ist nach dem ersten RESET nicht mehr ansprechbar. Braucht man das DOS noch, kann man z.B. BOOT auf eins lassen und DOSVEC anstatt CASINI ändern. Das klappt allerdings nicht, wenn ATMAS anwesend ist, da sich ATMAS auch in die RESET-Kette einklinkt.

## Basic-RESET

---

Wie Sie gesehen haben, ist ein RESET-Schutz für Maschinenprogramme gar nicht so schwer. Und wie sieht es mit Basic aus? Hier muß schon etwas tiefer in die Trickkiste gegriffen werden. Um ein Basic Programm nach einem RESET neu zu starten, muß immerhin ein RUN-Befehl simuliert werden. Auch dieses Problem ist nicht unlösbar: Wir schreiben nach einem RESET einfach RUN auf den Bildschirm und benutzen dann den "Forced-Read" Modus, mit dem sich Basic quasi selbst programmieren kann. Basic bekommt auf diese Art ein RUN, ganz so als ob es von Hand eingetippt worden wäre. Man muß nur darauf achten, daß der Auto-Modus vom gestarteten Basic wieder mit POKE 842,12 beendet wird. Sonst würden zukünftige INPUT nicht mehr funktionieren.

---

Ein derartiges Programm finden Sie in Listing 2a. Ab Zeile 100 beginnt das nach RESET gestartete Programm. Um ein beliebiges Basic-Programm gegen RESET zu schützen, brauchen Sie nur die Zeilen 20 bis 40 an dessen Anfang zu stellen.

Ein Assemblerlisting der DATA-Zeilen finden Sie in Listing 2b. Das Programm liegt in Page 6, daher dürfen Sie diesen Bereich im Programm nicht anderweitig verwenden. Die Adresse des JSR am Anfang ist nur ein Platzhalter und wird mittels zweier Poke im Basic-Teil durch den Inhalt von DOSINI ersetzt. Nach dem JSR wird zuerst der Bildschirm dunkel geschaltet, damit der Trick mit RUN im Verborgenen geschieht. Schließlich wird der IOCB Nr. 0 zur Ausgabe eines Strings eingerichtet, und dieser über CIO ausgegeben. Der Cursor wird wieder in die linke obere Ecke gestellt (wichtig!) und der "Forced-Read" Modus aktiviert.

Zusammen mit Basic funktioniert das dann so: In Zeile 20 wird geprüft, ob das Maschinenprogramm schon in Page 6 steht. Wenn nein, wird es aus den DATA dorthin transferiert. Anschließend wird der Inhalt von DOSINI in den JSR Befehl eingetragen (s.o.), und DOSINI selbst wird auf das ML-Programm in Page 6 gerichtet. Das darf aber nur einmal geschehen, sonst geht der ursprüngliche DOSINI-Wert verloren (Zeile 30). In Zeile 40 wird der "Forced-Read" Modus abgeschaltet, danach beginnt das Anwenderprogramm. Da im ML-Teil das Bild abgeschaltet wurde, sollte hier ein Poke 559,34 oder ein GRAPHICS Befehl erfolgen. Vollends narrensicher wird ein Programm, wenn man zusätzlich die Break Taste verriegelt. Im Beispiel ginge das mit der Zeile

```
105 POKE 16,64: POKE 53774,64
```

Aber Vorsicht: Nach einem RUN kommen auch Sie selbst nicht mehr an das Programm heran!

Der Neustart erfolgte im Beispiel durch das Anzapfen der DOSINI-Routine. Durch den Trick mit dem JSR bleibt die Initialisierung des DOS-Handlers D: nach wie vor erhalten; d.h. DOS steht auch nach einem RESET zur Verfügung. Mit diesem Trick kann man auch eine andere nützliche Anwendung erreichen. Reservierung von Speicherplatz.

### **Reservierter Platz**

---

Jeder kennt das Problem: Wohin mit Maschinenprogrammen. Zeichen Sätzen, PM-Speicher oder OS-Erweiterungen, wenn Basic verwendet wird? Page 6 ist dazu meist zu klein, die anderen Speicherbereiche sind ungeschützt und könnten überschrieben werden. Oft wird dazu die obere Speichergrenze mit RAMTOP (106, \$6A) herabgesetzt. Doch spätestens nach dem ersten RESET gibt es dann Probleme, da das OS den oberen Speicherbereich mit einem GRAPHICS 0 Bildschirm überschreibt. Lag ein Maschinenprogramm an dieser Stelle, so ist der Absturz nicht mehr weit. Es gibt auch keine Möglichkeit, die Einrichtung des GR.0 Bildschirms zu verhindern.

Listing 3a wartet hier mit einer eleganteren Lösung auf: Es erzeugt ein AUTORUN.SYS File, das die untere Speichergrenze (MEMLO) auf einen beliebigen Wert anhebt und somit einen geschützten Platz zwischen dem Ende von DOS und der neuen MEMLO-Grenze schafft. Dort können ohne Bedenken Daten und Programme aller Art aufbewahrt werden. Der Clou dabei ist, daß der Bereich auch nach einem RESET noch geschützt bleibt.

---

Tippen Sie Listing 3a ein, starten Sie es, legen Sie eine Diskette mit DOS ins Laufwerk 1 und geben Sie die gewünschte Untergrenze des Basic-Speichers (dezimal) ein. Nehmen Sie z.B. 16384 (\$4000 hex). Daraufhin wird das AUTORUN.SYS auf die Disk geschrieben. Booten Sie diese Disk erneut mit Basic und prüfen den freien Speicher mit FRE(0) - es ist erheblich weniger als sonst! Dafür haben Sie nun den Bereich von 7936 (\$1F00, MEMLO von DOS 2.5) bis 16384 (\$4000) reserviert. Drücken sie dann RESET und prüfen Sie noch einmal!

Wiederum wurde der Trick mit der Anzapfung von DOSINI verwendet (Assemblerlisting 3b). Nur wird diesmal kein RUN generiert, sondern der Zeiger MEMLO auf einen neuen Wert gesetzt. Wichtig ist, daß die DOS-Initialisierung vor der Veränderung des Zeigers erfolgt, da diese MEMLO auf das Ende von DOS setzt. Sehr häufig werden Sie solche Routinen bei OS-Erweiterungen finden (z.B. bei der RAMdisk aus CK 7/85), die damit etwas Basic-Speicher für sich abzweigen. Selbstverständlich können auch größere Bereiche geschützt werden, etwa wenn Platz für ein Hi-Res-Bild oder einen Zeichensatz benötigt wird. Nebenbei bemerkt: Dieses Verfahren funktioniert auch mit ACTION? hervorragend.

Peter Finzel

---

```

*****
* START NACH RESET
*
* Beispiel fuer Maschinenprogramme
*****
*
BOOT    EQU 9           ;2:Cass-Boot O.K.
CASINI  EQU $02         ;Vektor Init. nach RESET
SDLSTL  EQU $230        ;Zeiger auf D-List
VCOUNT  EQU $D40B       ;Rasterzeile
COLBK   EQU $D01A       ;Hintergrund
*
        ORG $0600
*
        LDA #2          ;Cass-Boot
        STA BOOT        ;vortaeuschen
        LDA #START:L    ;Start-Adresse
        STA CASINI      ;eintragen
        LDA #START:H
        STA CASINI+1
*
* Einsprung nach RESET
*
START    LDA #DLIST:L    ;Display-List
        STA SDLSTL      ;aktivieren
        LDA #DLIST:H    ;jetzt MSB
        STA SDLSTL+1
ENDLOS   LDA VCOUNT     ;und ein wenig
        STA COLBK       ;Farbe
        JMP ENDLOS
*
* einfache Displaylist
*
DLIST    DFB $70,$70,$70,$70,$70,$70
        DFB $70,$70,$70,$70,$70,$46
        DFB SCR:L,SCR:H,$41
        DFW DLIST
SCR      ASC % PROGRAMM LAEUFT... %

```

```
10 REM *****
11 REM * NEUSTART BEI RESET
12 REM *
13 REM * P. FINZEL 1986
14 REM *****
20 IF PEEK (1536)=32 THEN 30
21 FOR A=1536 TO 1591:READ D
22 POKE A,D:NEXT A
23 DATA 32,255,255,162,0,142,47,2
24 DATA 169,11,157,66,3,169,50,157
25 DATA 68,3,169,6,157,69,3,169,6
26 DATA 157,72,3,169,0,157,73,3,32
27 DATA 86,228,169,0,133,84,133,85
28 DATA 133,86,169,13,141,74,3,96
29 DATA 125,29,29,82,85,78
30 IF PEEK (1538)<>255 THEN 40
31 POKE 1537,PEEK (12)
32 POKE 1538,PEEK (13)
33 POKE 12,0:POKE 13,6
40 POKE 842,12
100 GRAPHICS 2+16:POSITION 0,5
110 ? #6; "PROGRAMM LAEUFT ...."
120 FOR I=0 TO 255:POKE 708,I:NEXT I
130 GOTO 120
```

```

*****
* BASIC-RUN BEI RESET
*
* PETER FINZEL 1986
*****
*
* IOCB-Konstante
*
CIOV      EQU  $E456
ICCOM     EQU  $342
ICBAL     EQU  $344
ICBAH     EQU  $345
ICBLL     EQU  $348
ICBLH     EQU  $349
ICAX1     EQU  $34A
CPBIN     EQU  11
*
* Sonstige Register
*
ROWCR5    EQU  $54          Cursor-Zeile
COLCR5    EQU  $55          Cursor-Spalte
SDMCTL    EQU  559          DMA-Register (5)
*
* Programm in PAGE 6
*
        ORG  $600
*
        JSR  $FFFF          Platz f. DOS-Init
        LDX  #0              'RUN' am oberen
        STX  SDMCTL          Bild aus
        LDA  #CPBIN          Rand des Bild-
        STA  ICCOM,X          schirmes aus-
        LDA  #RUN:L          geben.
        STA  ICBAL,X          Adresse des
        LDA  #RUN:H          Strings in IOCB
        STA  ICBAH,X
        LDA  #RUNEND-RUN      Laenge
        STA  ICBLL,X
        LDA  #0
        STA  ICBLH,X
        JSR  CIOV            CIO aufrufen
        LDA  #0              Cursor nach
        STA  ROWCR5          links oben
        STA  COLCR5          stellen
        STA  COLCR5+1
        LDA  #$0D             Forced-Read
        STA  ICAX1           Eingabe-Modus
        RTS
*
RUN      DFB  $7D,$1D,$1D      Clear Screen
        ASC  "RUN"            Cursor 2x unten
RUNEND   EQU  *

```



```

3
100 REM * MEMLO-VERSCHIEBUNG
110 REM * ERZEUGT AUTORUN.SYS
120 REM
130 REM * P. FINZEL
140 ? "Untere Speichergrenze bei:";:INPUT MEMLO
150 MH=INT(MEMLO/256)
160 ML=MEMLO-MH*256
170 DIM D$(80):OPEN #1,8,0,"D:AUTORUN.SYS"
180 ? "AUTORUN.SYS wird generiert...":? :FLAG=0
190 READ D$:READ P:IF D$="*" THEN 290
200 S=0:? "*";
210 FOR I=1 TO LEN(D$) STEP 2
220 H=ASC(D$(I,I))-48:L=ASC(D$(I+1,I+1))-48
230 D=(H-(H>9)*7)*16+L-(L>9)*7:S=S+D
240 IF FLAG=1 THEN 270
250 IF I=13 THEN D=ML
260 IF I=15 THEN D=MH:FLAG=1
270 PUT #1,D:NEXT I:IF S=P THEN 190
280 ? :? :? "DATENFEHLER IN ZEILE
";PEEK(183)+PEEK(184)*256:CLOSE #1:STOP
290 REM * FILE SCHLIESSEN
300 CLOSE #1
310 ? :? :? "AUTORUN.SYS ordnungsgemaess erzeugt!"
320 END
1100 DATA FFFF00062706004020FFFFAD00068DE702AD01068D,2041
1110 DATA E80260A50C8D0306A50D8D0406A902850CA906850D,1623
1120 DATA 4C050600FFFFE202E3021206,1078
1130 DATA *,0

```

```

*****
* Speicherplatz-Reservierung m. MEMLO
* Aufzeichnen: als AUTORUN.SYS
* von $600 bis $627
* und $2E2 bis $2E3 (mit Append!)
* Assembler: ATMAS-II
*****
*
DOSINI EQU $0C;DOS-Init nach RESET
MEMLO EQU $2E7 ;untere Speichergrenze
*
    ORG $600
*
LOWRAM DFW $4000 ;neues MEMLO (wird ersetzt)
*
* Einsprung nach RESET
* wird in die DOS-Initialisierung
* eingebunden
*
NEUINI JSR $FFFF ;Platz fuer DOSINI
KALT LDA LOWRAM ;neue MEMLO-Adresse
    STA MEMLO ;eintragen (waehrend
    LDA LOWRAM+1 ;RESET!)
    STA MEMLO+1
    RTS ;das war's!
*
* Einbindung der Routine in den
* RESET-Ablauf
*
START LDA DOSINI ;DOS-Init in leeren
    STA NEUINI+1 ;JSR eintragen
    LDA DOSINI+1 ;jetzt MSB
    STA NEUINI+2
    LDA #NEUINI:L ;neues DOSINI
    STA DOSINI ;eintragen
    LDA #NEUINI:H
    STA DOSINI +1
    JMP KALT ;MEMLO vorbesetzen
*
* File-Init Adresse fuer AUTORUN.SYS
*
    ORG $2E2
    DFW START

```

---