Assembler ist keine Alchimie — Teil 11

Noch einmal geht es um Unterbrechungen. Wie geht unser Computer bei den verschiedenen Möglichkeiten vor? Wo kann man selbst einhaken und wie schreibt man eigene Unterbrechungsprogramme? Anhand von zwei Beispielen lernen Sie in dieser Folge, wie man den Ausstieg aus einem Programm verhindert und wie man die Rasterzeilenunterbrechung für eigene Programme einsetzt.

In der letzten Folge haben Sie erfahren, wie unsere CPU mit Unterbrechungen umgeht, welche Sorten von Unterbrechungen es gibt, welches Instrumentarium die Assembler-Sprache zur Behandlung dieser speziellen Technik bietet, und Sie kennen die »primären« Quellen für eine Unterbrechung. Diesmal wollen wir analysieren, wie die Unterbrechungen softwaremäßig bearbeitet werden, um Wege zu finden, die uns auf möglichst einfache Weise Eingriffe erlauben.

Der normale Verlauf eines IRQ

Neulich hatten wir bereits festgestellt, daß eine IRQ-Anforderung (nach dem Retten des Programmzählers und des Prozessorstatus-Registers, sowie dem Setzen der I-Flagge) den Inhalt des Vektors $FFFE/FFFF in den Programmzähler holt. Dort steht die Adresse $FF48 (dez. 65352) und deshalb startet nun das dort im ROM verankerte Programm, welches wir uns nun im einzelnen ansehen werden (alle Adressen als Dezimalzahlen, in Bild 1 finden Sie das Flußdiagramm dazu).

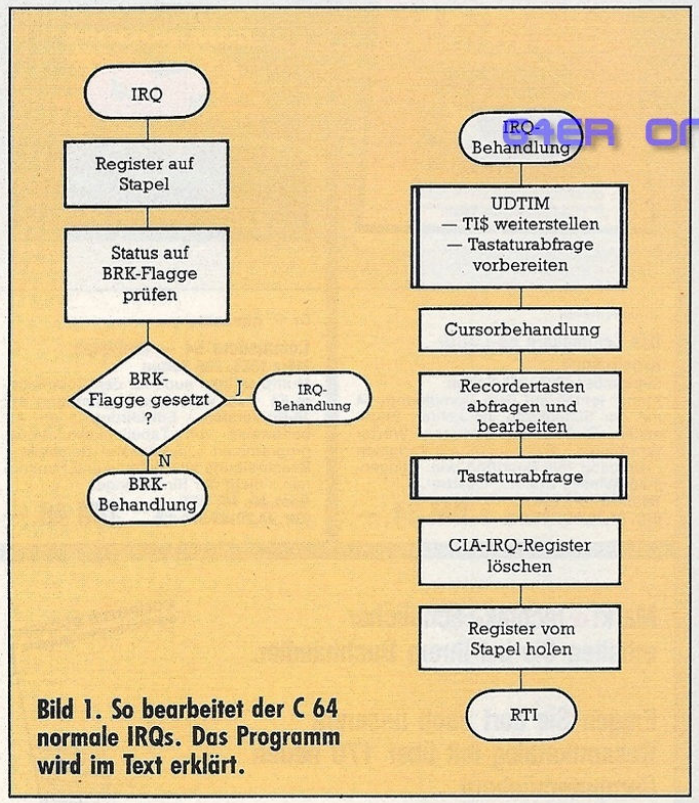

Bild 1. So bearbeitet der C 64 normale IRQs. Das Programm wird im Text erklärt.

```
65352   PHA     Zunächst werden der Akku und die
        TXA     Register X und Y auf den Stapel ge-
        PHA     schoben
        TYA
        PHA
```

Trickreich sind die beiden folgenden Befehle, mit denen das zu Beginn durch die CPU gerettete Statusregister gelesen wird:

```
        TSX         Stapelzeiger ins X- Register
        LDA 260,X   Einladen des Status-Registers
```

Nun wird geprüft, ob die BRK-Flagge gesetzt ist. Wenn das der Fall ist, dann ist der Auslöser ein BRK gewesen, ansonsten ein IRQ:

```
        AND #16     Isolieren der BRK-Flagge
        BEQ 65368   Wenn keine BRK-Flagge, dann über-
                    springen des nächsten Befehls.
65365   JMP (790)   Falls BRK
65368   JMP (788)   Falls IRQ
```

Den vorletzten Sprungbefehl werden wir bei der BRK-Behandlung verfolgen. Interessant für uns ist jetzt der indirekte Sprung bei 65368. Der Vektor 788/789 ($314/315) liegt im RAM! Damit können wir ihn auf eigene Routinen verstellen. Genau hier ist der Ansatzpunkt für nahezu alle Eingriffe in die Unterbrechungsbehandlung. Der voreingestellte Wert in diesem Vektor ist die Adresse 59953 ($EA31). Das dort angesiedelte Programm wird im Normalfall 60mal in der Sekunde ausgeführt:

```
59953   JSR 65514   Das ist ein Kernal-Sprungbefehl zur
                    Routine UDTIM bei 63131.
```

In diesem Unterprogramm wird zuerst die Uhr TI$ weitergestellt und dann die Tastaturabfrage vorbereitet.

```
59956 bis   In diesem Programmteil erfolgt die
60000       Cursorbehandlung.
60001 bis   Anschließend wird abgefragt, ob eine
60026       Recordertaste gedrückt ist und ent-
            sprechende Flaggen bearbeitet.
60027   JSR 60039   Dieses Unterprogramm dient zur
                    Tastaturabfrage.
```

Auch in dieser Routine tritt übrigens ein indirekter Sprung nach einem RAM-Vektor auf (655/656 = $28F/290), der normalerweise auf 60232 zeigt, aber auch auf eine eigene Routine verbogen werden könnte.

Enthalten in der Tastaturabfrage ist auch die Überprüfung der RUN/STOP-Taste, die aber nur zusammen mit den in dem UDTIM-Aufruf voreingestellten Flaggen funktioniert. Deshalb wird das Abschalten der RUN/STOP-Taste im allgemeinen dadurch durchgeführt, daß man den IRQ-Vektor auf 59956 stellt und damit den ersten JSR-Befehl überspringt. Allerdings wird auf diese Weise auch die TI$-Uhr nicht weitergestellt.

```
60030   LDA 56333   Das ist das Unterbrechungs-
                    Kontrollregister des IRQ-CIA, das
                    hier durch Auslesen gelöscht wird.
```

Den Abschluß der IRQ-Routine bildet nun noch das Zurückschreiben der Register:

```
60033   PLA     Zurückholen des
        TAY     Y- und
        PLA
        TAX     des X-Registers
        PLA     sowie des Akku.
60038   RTI     Damit kehrt der Computer zu dem
                durch den IRQ unterbrochenen
                Programm zurück.
```

Somit hätten wir's. Nun können wir je nach Bedarf entscheiden, welche von diesen Servicetätigkeiten wir bei einem eigenen IRQ-Programm brauchen: Die Uhr TI$, die Cursorbehandlung, die Abfrage der Recordertasten und die Tastaturabfrage.

Sehen wir uns nun an, was geschieht, wenn ein BRK-Kommando der Auslöser war.

BRK-Unterbrechung

Wir hatten vorhin am Scheideweg zwischen IRQ und BRK den letzteren links liegen gelassen. Normalerweise verwendet man beim Programmieren in Assembler ja ein Software-Instrument wie zum Beispiel den SMON, der so gebaut ist, daß der BRK-Vektor, welchen wir vorhin kennengelernt haben ($316/317 = 790/791) auf die Registeranzeige weist. Was geschieht eigentlich, wenn der BRK-Vektor unverändert bleibt, so also, wie er im Einschaltzustand des Computers vorliegt?

Dann zeigt er auf die Adresse 65126 ($FE66), wo ein Teil der NMI-Routine zu finden ist (Siehe auch das Flußdiagramm in Bild 2):

Bild 2. Auf diese Weise verläuft ein unvorhergesehener BRK im Sande

65126 JSR 64789 Sprung ins Programm RESTOR, in dem alle Vektoren (788-819) gemäß einer ROM-Liste auf ihre Ausgangswerte gesetzt werden.

JSR 64931 Sprung in das Programm I/O-RESET.

In diesem Programm werden die beiden CIAs auf die Anfangswerte gestellt.

JSR 58648 Sprung in ein Programm, welches zuerst den VIC-II-Chip initialisiert, dann einen Bildschirmeditor-RESET durchführt. Nach Beenden dieser Routine ist der Bildschirm gelöscht.

JMP (40962)

Mit diesem indirekten Sprung ist die BRK-Unterbrechung beendet. Man sieht aber jetzt schon deutlich, daß es sich hier nicht um eine Unterbrechung im eigentlichen Sinn handelt, vielmehr um einen Abbruch. In 40962/40963 steht die Adresse des Basic-Warmstarts (58235). Danach befindet sich der Computer im READY-Zustand in der Eingabe-Warteschleife.

Das Zurückholen der Register und ein RTI erübrigt sich hier, weil ohnehin viele Werte aus dem unterbrochenen Programm inzwischen weitgehend zerstört sind und alle Unterbrechungskontrollregister (CIAs und VIC-II-Chip) neu belegt wurden. Ein unkontrollierter BRK hat also recht fatale Folgen!

Was macht ein NMI?

Wenden wir uns nun der Firmware zu, die zur Bearbeitung eines NMI vorgesehen ist (Dazu sehen Sie sich bitte in Bild 3 das Flußdiagramm an).

In der letzten Folge erfuhren wir, daß auch für diese Unterbrechung am Ende des Speichers ein Vektor vorhanden ist, nämlich $FFFA/FFFB (65530/65531). Dort steht die Adresse 65091 ($FE43), die nun in den Programmzähler gelangt und damit startet das folgende Programm:

65091 SEI Unterbrechungen niedrigerer Priorität werden gesperrt.

JMP (792)

Das ist nun wieder ein für uns sehr interessanter Vektor 792/793 ($318/319), der — weil er im RAM-Bereich liegt — verstellbar ist. Genau das haben wir am Ende der letzten Folge getan mittels des M-Kommandos von SMON um den NMI zu testen, den wir mit der RESTORE-Taste ausgelöst haben. Der voreingestellte Wert in diesem Vektor ist die Adresse 65095 ($FE47), also direkt der nächste Befehl nach dem indirekten Sprungbefehl.

65095 PHA Ebenso wie vorhin beim IRQ werden
TXA hier die Inhalte des Akku und der
PHA Register auf den Stapel geschoben.

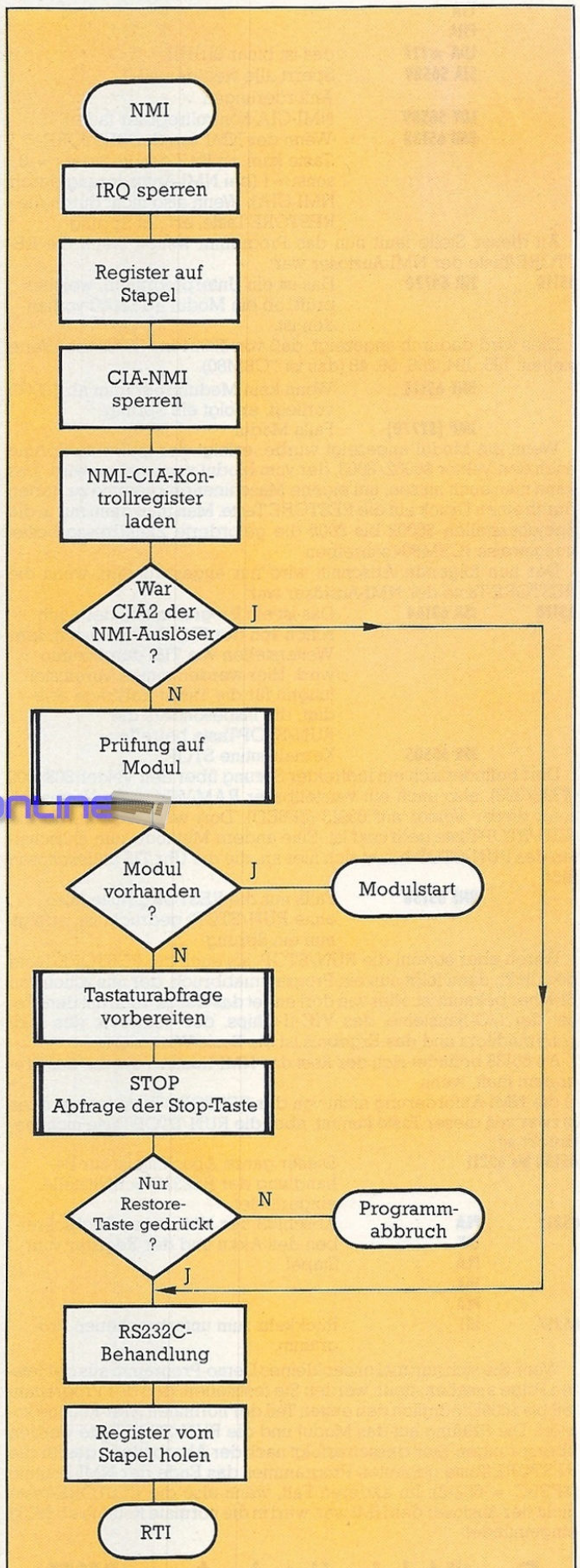

Bild 3. Flußdiagramm zum Ablauf einer NMI-Unterbrechung

```
TYA
PHA
LDA #127        das ist binär 01111111.
STA 56589       Sperrt alle weiteren NMI-
                Anforderungen
LDY 56589       NMI-CIA Kontrollregister laden.
BMI 65138       Wenn der NMI von der RESTORE-
                Taste kam, ist Bit 7 des Registers = 0,
                sonst =1 (bei NMI-Anforderung durch
                NMI-CIA). Wenn also nicht durch die
                RESTORE-Taste, erfolgt Sprung.
```

An dieser Stelle läuft nun das Programm weiter, wenn die RESTORE-Taste der NMI-Auslöser war:

```
65110   JSR 64770       Das ist ein Unterprogramm, welches
                        prüft, ob ein Modul ab $8000 vorhan-
                        den ist.
```

Dies wird dadurch angezeigt, daß von $8004 bis $8008 die Werte stehen: 195, 194, 205, 56, 48 (das ist "CBM80).

```
        BNE 65118       Wenn kein Modulprogramm ab $8000
                        vorliegt, erfolgt ein Sprung.
        JMP (32770)     Falls Modul.
```

Wenn ein Modul angezeigt wurde, erfolgt der indirekte Sprung nach den Vektor $8002/8003, der vom Modul vorgegeben wird. Das kann man auch nutzen, um eigene Maschinenprogramme zu starten durch einen Druck auf die RESTORE-Taste. Man muß dann nur in die Speicherstellen $8002 bis 8008 die geforderte Zieladresse beziehungsweise »CBM80« schreiben.

Der nun folgende Abschnitt wird nur angesprungen, wenn die RESTORE-Taste der NMI-Auslöser war:

```
65118   JSR 63164       Das ist ein Programmteil, der auch
                        schon von der IRQ-Routine (nach dem
                        Weiterstellen von TI$) durchlaufen
                        wird. Hier werden einige Voreinstel-
                        lungen für die Tastaturabfrage erle-
                        digt, die insbesonders die
                        RUN/STOP-Taste betreffen.
        JSR 65505       Kernalroutine STOP.
```

Dort befindet sich ein indirekter Sprung über den Vektor 808/809 ($328/329), also auch ein verstellbarer RAM-Vektor. Im Normalfall zeigt dieser Vektor auf 63213 ($F6ED). Dort wird geprüft, ob die RUN/STOP-Taste gedrückt ist. Eine andere Methode zum Ausschalten des RUN/STOP bietet sich hier an, die die Uhr TI$ ungeschoren läßt.

```
        BNE 65138       Falls nur die RESTORE-Taste (also
                        ohne RUN/STOP) gedrückt ist, erfolgt
                        nun ein Sprung.
```

Waren aber sowohl die RUN/STOP- als auch die RESTORE-Taste gedrückt, dann folgt nun ein Programmabbruch, der uns schon von BRK her bekannt ist. Hier wie dort endet das Ganze dann mit dem Reset der I/O-Bausteine, des VIC-II-Chips, der Vektoren, des Bildschirmeditors und das Ergebnis ist ein Basic-Warmstart.

Ab 65138 befindet sich der Rest der NMI-Routine, auf die das Programm läuft, wenn
1) die NMI-Anforderung nicht von der RESTORE-Taste kommt oder
2) zwar von dieser Taste kommt, aber die RUN/STOP-Taste nicht gedrückt ist.

```
65138 bis 65211         Dieser ganze Abschnitt ist zur Be-
                        handlung der RS232C-Schnittstelle
                        eingerichtet.
65212   PLA             Abschluß des NMI durch Rückschrei-
        TAY             ben des Akku und der Register vom
        PLA             Stapel
        TAX
        PLA
65217   RTI             Rückkehr zum unterbrochenen Pro-
                        gramm.
```

Wenn Sie sich nun mal unser kleines Demo-Programm aus der letzten Folge ansehen, dann werden Sie feststellen, daß der Programmteil bis $600E lediglich den ersten Teil der normalen NMI-Routine kopiert. Die Prüfung auf das Modul und die RUN/STOP-Taste werden übersprungen. Statt dessen erfolgt nach der Abarbeitung des für die RESTORE-Taste gebauten Programmes das Ende der NMI-Routine ($FEBC = 65212). Im anderen Fall, wenn also die RESTORE-Taste nicht der Auslöser des NMI war, wird in die normale Routine ab 65138 eingemündet.

Eigentlich keine Unterbrechung: RESET

Weil wir alle Unterbrechungen hier bearbeiten wollen, soll auch der RESET angesprochen werden. Es handelt sich dabei aber nicht um eine Unterbrechung im bisher definierten Sinn. Mir fällt allerdings kein Platz ein in dieser Serie, wo der RESET besser hinpassen würde. Ähnlich wie bei NMI und IRQ wird auch hier ein Vektorinhalt in den Programmzähler geladen, der in den höchsten Speicheradressen zu finden ist (Auch hierzu wieder ein Flußdiagramm in Bild 4).

Dieser Vektor liegt in $FFFC/FFFD. Der Inhalt ist die Adresse 64738 ($FCE2) und genau dort geht das Programm dann weiter:

```
64738   LDX #255        Im ersten Teil wird der
                        Stapelspeicher initialisiert.
        SEI             Verhindern von IRQ
        TXS             Stapelzeiger auf $FF
        CLD             Dezimal-Modus ausschalten (falls er
                        eingeschaltet war).
        JSR 64770       Das ist wieder das Unterprogramm,
                        das auf ein Modul prüft.
```

Hier ergibt sich die Möglichkeit, auch beim RESET einzugreifen, indem man die Kennung CBM80 an die abgefragten Orte packt.

```
        BNE 64751       Falls kein Modul, erfolgt Sprung.
64748   JMP(32768)
```

Dieser indirekte Sprung erfolgt nach dem Vektorinhalt von $8000/8001 = 32768/32769. Das ist ein anderer Vektor als wir ihn vorhin beim NMI hatten (dort war es $8002/8003 = 32770/32771). So kann ein anderer Programmteil angesteuert werden als durch den NMI, was übrigens auch dringend erforderlich ist, weil der Stapelzeiger zerstört wurde.

```
64751                   Hier läuft das Programm weiter, falls
                        keine Modulkennung erkannt wurde.
```

Der ganze Rest dient dem Versetzen des Computers in den Einschaltzustand. Allerdings bin ich davon überzeugt, daß noch irgendein Unterschied bestehen muß zwischen dem einfachen Aus- und wieder Anschalten des Computers und einem RESET. Es hat sich nämlich bei einigen Programmen gezeigt, daß sie nach einem RESET fehlerhafte Verläufe nehmen können, was nach einem totalen Aus- und wieder Anschalten nicht zu beobachten war. Der Grund für diesen Unterschied liegt (für mich) noch im Dunkel. Vielleicht weiß das ja jemand von Ihnen. Dann schreiben Sie doch mal!

In der nächsten Folge werden wir der Sache mit dem Modulstart noch etwas weiter auf den Grund gehen und auch ein interessantes Programm dazu entwickeln. (Heimo Ponnath/gk)

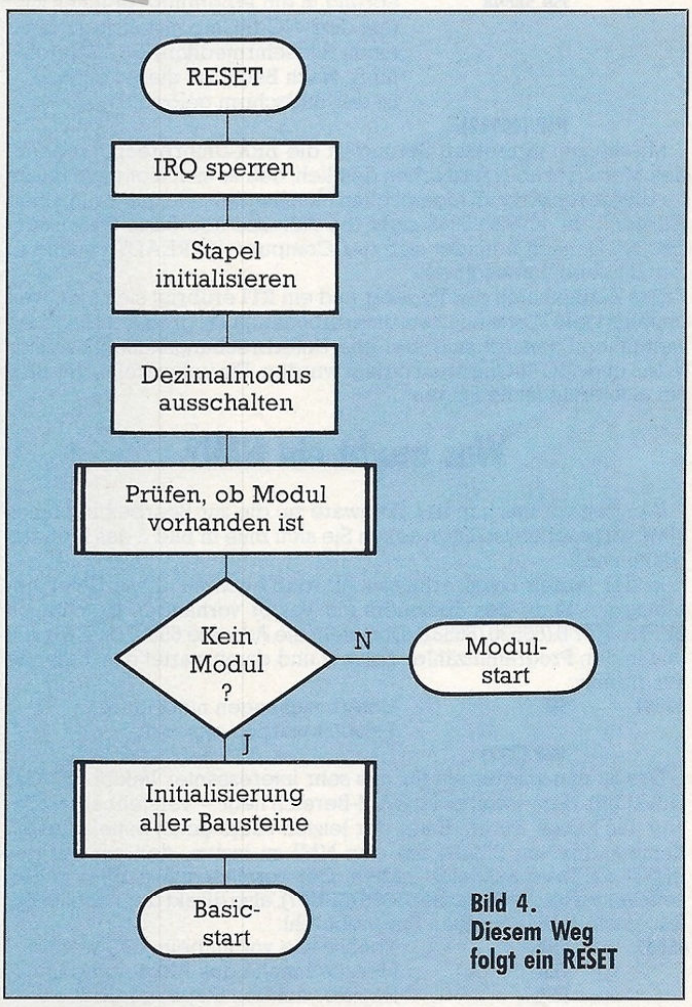

Bild 4. Diesem Weg folgt ein RESET