Assembler ist keine Alchimie — Teil 12

Wie macht man Programme Reset-fest? Außerdem schauen wir dem »Rasterzeilen-Interrupt« etwas genauer auf die Finger und entwickeln ein Programm dazu.

Die Sache mit dem Modulstart

Sowohl beim RESET als auch beim NMI haben wir festgestellt, daß der Modulstart-Bereich ab $8000 eine besondere Rolle spielt. In Bild 5 finden Sie nochmal zusammengefaßt, was sich dort findet wenn ein Modul vorhanden ist.

Wir wollen im folgenden Beispielprogramm (Listing 1) ein Modul simulieren, indem wir den SMON mittels des RESET anspringen. Der NMI — also die RUN/STOP-RESTORE-Tastenkombination — soll dabei wirkungslos gemacht werden.

Bild 6 zeigt ein Flußdiagramm dieses Beispielprogrammes:

Achten Sie bitte darauf, daß Sie nach dem Eintippen des Programmes abspeichern und — natürlich — daß die SMON-Version ab $C000 im Speicher vorliegt. Mit SYS 24576 starten Sie unser Programm, in dem durch diesen SYS-Befehl zunächst nach $8000/1 die Startadresse einer neuen RESET-Serviceroutine geschrieben wird und nach $8002/3 die der neuen NMI-Routine. Außerdem wird die Modulkennung in die vorgeschriebenen Speicherplätze eingetragen. Wenn Sie nun mal die RESTORE-Taste — oder RUN/STOP und RESTORE — drücken, passiert offensichtlich nichts. Das liegt daran, daß unser Programm lediglich die auf den Stapel gelegten Register wieder zurückholt und aus der Unterbrechung mit RTI ins normale Geschehen zurückkehrt.

Haben Sie einen RESET-Taster eingebaut? Dann drücken Sie doch mal drauf. Zunächst erkennen Sie den normalen RESET-Verlauf. Dann meldet sich aber nicht wie gewohnt die Nachricht CBM-Basic..., sondern der SMON mit einer Registeranzeige. Das RESET-Programm ab $602E folgt dem Firmware-Programm. Lediglich der letzte Sprungbefehl ist anders und führt statt ins Basic in den SMON. Der SMON wird fehlerfrei funktionieren... solange Sie nicht versuchen, mit dem X-Kommando wieder ins Basic zurückzukehren. Dann wird Unsinn passieren, denn auf einen Start mittels RESET ist der SMON nicht gefaßt gewesen und in den Speicherstellen, die sonst eine Rückkehradresse enthalten, befindet sich nichts Sinnvolles. Es ist daher auch nicht möglich, den SMON wieder zu verlassen — außer durch Speicherstellenmanipulationen oder die Notbremse: Aus- und wieder Einschalten. Auf diese Weise (und mittels eines AUTOSTART) sichern sich Softwarehäuser manchmal gegen unbefugtes Kopieren ihrer Programme.

Nutzung der Unterbrechungen

Sowohl was die Hardware als auch die Firmware für die Unterbrechungsbehandlung angeht, haben wir nun einen guten Überblick gewonnen. Es ist jetzt an der Zeit, daß wir uns ansehen, auf welche Weise man dieses Reservoir an vielfältigen Möglichkeiten für sich nutzen kann. Dazu soll uns ein Überblick dienen:

I) Auslösung der Unterbrechung durch Hardware-Einwirkungen.

Da hätten wir beispielsweise den Userport oder den Expansion-Port, über die wir per CIAs Unterbrechungen anfordern können. Um es gleich zu sagen: Damit werden wir uns nicht auseinandersetzen. Meine Kenntnisse auf diesem Gebiet sind zu dünn. Aber vielleicht verstehen Sie das auch mal als Aufforderung, Ihre Versuche dazu anderen zu offenbaren? Also: Schreiben Sie doch mal!

II) Unterbrechungsauslösung per Software:

Damit haben wir immer noch ein weites Feld von Möglichkeiten vor uns:

IIa) Vorgesehene Nutzungen des IRQ

— mittels des VIC-II-Chips.

Da können wir uns auf den Rasterzeileninterrupt, die Sprite/Hintergrund- oder die Sprite/Sprite-Kollision stützen.

— oder mit Hilfe des CIA1

Da ist es vor allem der 60mal pro Sekunde auftretende Timer A-Unterlauf, der uns interessieren soll.

IIb) Vorgesehene Nutzungen des NMI

— CIA2: Läßt man die RS232C-Schnittstellenbehandlung außer acht, dann gibt es keine vorgesehene Nutzung.

— RESTORE: Zusammen mit der RUN/STOP-Taste kann man die vorgegebene Routine verändern, wie wir es schon in einigen Beispielen gezeigt haben.

Wir können außerdem noch unterscheiden zwischen Nutzungen, die periodisch stattfinden sollen (zum Beispiel eine spezielle Tastaturabfrage) und solchen, die stochastisch (= zufallsabhängig) oder willkürlich erfolgen (zum Beispiel Drücken der RESTORE-Taste). Beides ist auch durchführbar bei:

IIc) Nicht vorgesehene Nutzung der Unterbrechungen.

Da bietet sich vor allem der meistens völlig brach liegende CIA2 an mit seinen beiden Timern und der Alarmfunktion.

Wenn Sie aber erst einmal vertraut sind mit der Unterbrechungs-Programmierung und auch etwas Zeit zum Tüfteln investieren, finden Sie bestimmt noch eine ganze Menge weiterer Möglichkeiten.

Bei mehreren gleichartigen Unterbrechungsanforderungen (zum Beispiel IRQs) muß noch ein Weg gefunden werden, wie zwischen den dann vielleicht anfallenden unterschiedlichen Service-Routinen differenziert werden kann. Denkbar wären beispielsweise Aufgabenstellungen wie:

Jeder 3. Timer-IRQ soll den Joystick abfragen, oder RESTORE + h soll den Hilfsbildschirm zeigen, RESTORE + z soll den aktuellen Bildschirm wieder restaurieren, etc.

Sie sehen, eine große Menge Arbeit wartet auf uns. Nicht zu allen Möglichkeiten werde ich hier Beispielprogramme zeigen. Außerdem dürfen die dann auch nicht zu undurchsichtig sein und man sollte möglichst den Erfolg eines solchen Demo-Programmes auf dem Bildschirm erkennen können. Trotzdem hoffe ich, daß die nachfolgend und noch in der nächsten Folge gezeigten Programmlösungen ausreichen, Ihnen die Unterbrechungs-Behandlung mit eigenen Routinen durchschaubar zu machen. Ich will Ihnen aber nicht verschweigen, daß auch mir noch längst nicht alle Geheimnisse der Unterbrechungsprogrammierung offenbar geworden sind. Oft finde ich mich unversehens in Programm-Sackgassen wieder. Das soll Ihnen als kleiner Trost dienen, wenn Sie mal nach dem 1001. Absturz müde und mit rauchendem Kopf vor Ihrem Commodore-Ungeheuer sitzen.

Ein Programm zum VIC-II-IRQ

Sehr schöne Effekte lassen sich durch eine periodische IRQ-Anforderung per Rasterzeileninterrupt mittels des VIC-II-Chip erzielen. Deshalb ist sowas auch ein beliebtes Objekt für Demos von Unterbrechungsprogrammen. Als Ziel setzen wir uns, einen Bildschirm zu konstruieren, dessen Rahmen in allen Farben schillert.

Leser der Grafikserie werden diese Möglichkeit des VIC-II-Chip schon kennen: Man kann dem Kathodenstrahl, der über den Monitor huscht, um das Bild zu erzeugen, über zwei Register folgen, die Rasterregister, wo jede Rasterzeile mitgezählt wird. Ohne an dieser Stelle allzusehr in die Einzelheiten einzugehen, soll hier nur bemerkt werden, daß die Numerierung dabei etwa von 0 bis 280 geht, weil

```
PROGRAMM 1

,6000  A9 2E     LDA #2E
,6002  8D 00 80  STA 8000
,6005  A9 60     LDA #60
,6007  8D 01 80  STA 8001
,600A  A9 41     LDA #41
,600C  8D 02 80  STA 8002
,600F  A9 60     LDA #60
,6011  8D 03 80  STA 8003
,6014  A9 C3     LDA #C3
,6016  A2 C2     LDX #C2
,6018  A0 CD     LDY #CD
,601A  8D 04 80  STA 8004
,601D  8E 05 80  STX 8005
,6020  8C 06 80  STY 8006
,6023  A9 38     LDA #38
,6025  A2 30     LDX #30

,6027  8D 07 80  STA 8007
,602A  8E 08 80  STX 8008
,602D  60        RTS
-------------------------
,602E  8E 16 D0  STX D016
,6031  20 A3 FD  JSR FDA3
,6034  20 50 FD  JSR FD50
,6037  20 8A FF  JSR FF8A
,603A  20 5B FF  JSR FF5B
,603D  58        CLI
,603E  4C 00 C0  JMP C000
-------------------------
,6041  68        PLA
,6042  A8        TAY
,6043  68        PLA
,6044  AA        TAX
,6045  68        PLA
,6046  40        RTI
```

Programm 1. Simulation eines Moduls

Speicherplatz ($)	8000	8001	8002	8003	8004	8005	8006	8007	8008
Inhalt	LSB	MSB	LSB	MSB	C	B	M	8	0
	RESET-Vektor		NMI-Vektor						

Bild 5. Diesen Inhalt müssen die Speicherstellen $8000 bis $8008 haben, damit ein Modulstart stattfindet

auch der Rahmen und nicht sichtbare Teile des Bildschirmes vom Strahl überstrichen werden. Wo das Textfeld anfängt, ist von Monitor zu Monitor (oder Fernseher) etwas unterschiedlich. Bei mir beginnt es oben in Rasterzeile 50 und endet unten bei Zeile 248. Sollten die im Beispielprogramm 2 (Listing 2) nachher voreingestellten Rand-

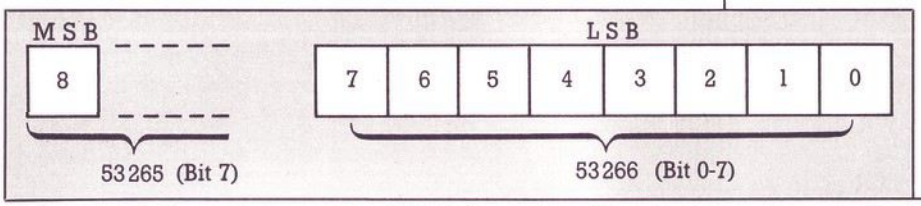

Bild 7. So sieht das 9-Bit-Register im VIC-II-Chip aus, welches die Rasterzeilen mitzählt

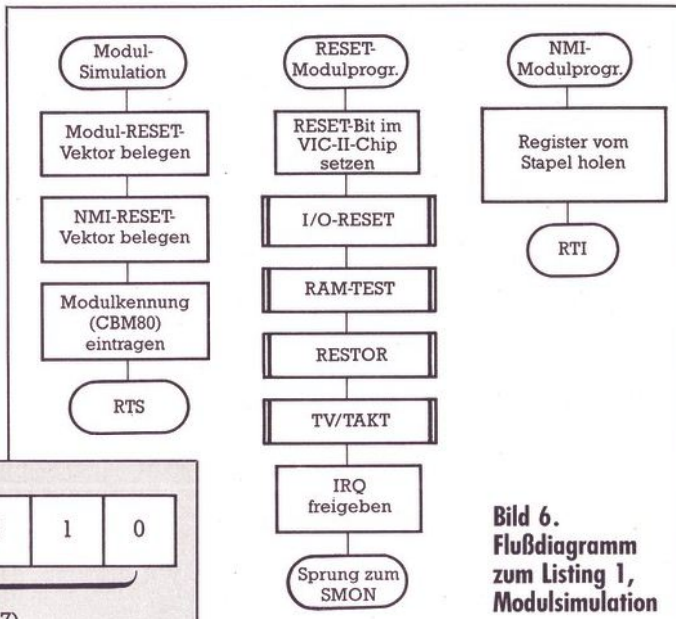

Bild 6. Flußdiagramm zum Listing 1, Modulsimulation

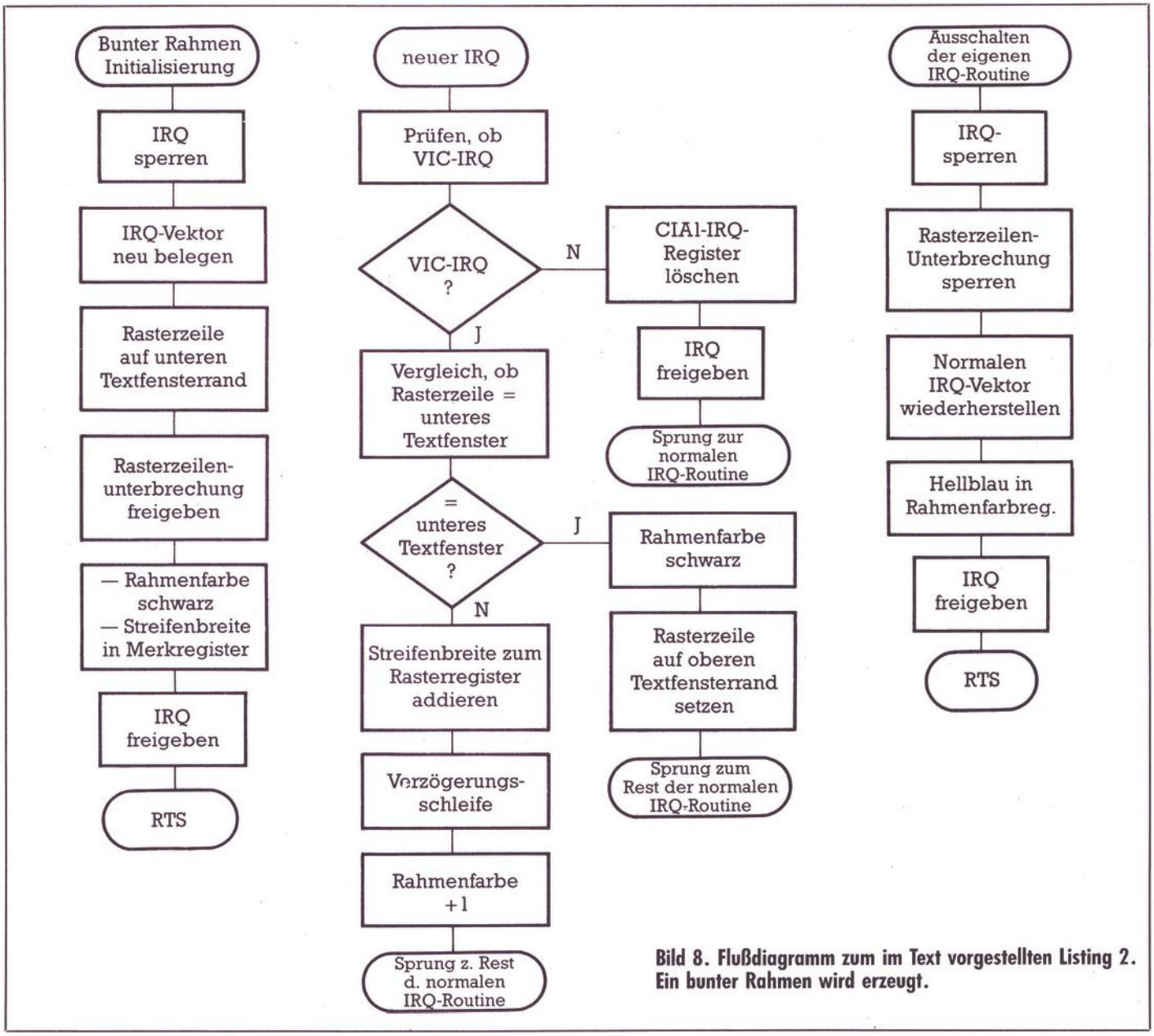

Bild 8. Flußdiagramm zum im Text vorgestellten Listing 2. Ein bunter Rahmen wird erzeugt.

werte bei Ihnen also anders sein, können Sie sie durch einige später noch angegebenen POKEs ändern. Die beiden Rasterzeilenregister sind:

$D012 (53266)
$D011 (53265)

Von $D011 allerdings ist nur das Bit 7 als msb der Rasterzeilenzahl für uns von Bedeutung. Bild 7 soll diese Belegung deutlich machen:

Das Interessante an diesen Registern ist nun, daß man auch in sie schreiben kann. Die auf diese Weise festgelegte Rasterzeile ist dann der Auslöser des IRQ, falls dieser im Interrupt-enable-Register $D01A freigegeben wurde (das kennen wir noch aus der letzten Folge).

Damit kann also unsere primäre Unterbrechungsquelle (der VIC-II-Chip) programmiert werden. Halten wir die zwei Schritte dazu nochmal fest:

1) Rasterzeile festlegen, bei der ein IRQ ausgelöst werden soll, durch Einschreiben in die Register $D012 und Bit 7 von $D011.
2) Freigeben des Rasterzeileninterrupts durch Einschreiben von 1000 0001 in das Interrupt-enable-Register $D01A.

Der nächste Schritt betrifft die Bearbeitung des IRQ durch die CPU. Wie wir vorhin sahen, springt das Programm beim IRQ mittels eines indirekten Sprunges, der auf den Vektor 788/9 ($314/5) zugreift. Dieser Vektor muß nun auf die eigene Routine verbogen werden, also:

3) Vektor $314/5 auf die IRQ-Service-Routine richten.

Damit wären alle Vorbereitungen getroffen. Der Rest liegt nun ganz bei uns — beziehungsweise bei dem von uns zu schreibenden Service-Programm. Als Bild 8 finden Sie ein Flußdiagramm unseres Beispielprogrammes 2.

Gehen wir nun an die Realisierung. Zunächst also die Initialisierung, die wir bei $6000 (also durch SYS 24576 zu starten) beginnen lassen:

6000	**SEI**	Sperren von IRQs
Schritt 3:		
6001	**LDA #$28**	LSB der IRQ-Routine
6003	**STA 0314**	in IRQ-Vektor-LSB
6006	**LDA #$60**	MSB der IRQ-Routine
6008	**STA 0315**	in IRQ-Vektor-MSB
Schritt 1:		
600B	**LDA #$F8**	Rasterzeile, bei der das Textfenster endet. Von da an soll der Rahmen schwarz sein.
600D	**STA D012**	in Rasterzeilen-Register (LSB) schreiben.
6010	**LDA D011**	Register mit dem msb des Rasterzeilenzählers
6013	**AND #$7F**	0111 1111 löscht das Bit7
6015	**STA D011**	Zurückschreiben. Damit ist die Rasterzeile, die den IRQ auslösen soll, festgelegt.
Schritt 2:		
6018	**LDA #$81**	1000 0001 wird nun
601A	**STA D01A**	ins IRQ-enable-Register geschrieben, um den Rasterzeilen-IRQ zuzulassen.
Festlegen einiger Startwerte:		
601D	**LDA #$00**	Farbe schwarz
601F	**STA D020**	in Rahmen schreiben
6022	**LDA #$04**	Streifenbreite in
6024	**STA 02**	Merkregister schreiben.
6026	**CLI**	IRQ freigeben
6027	**RTS**	Ende der Initialisierung.

Von nun an laufen alle IRQs über unsere eigene Routine, die bei $6028 beginnt.

Zunächst müssen wir prüfen, ob die Unterbrechung vom VIC-II-Chip kommt oder vom CIA1:

6028	**LDA D019**	IRQ-Request-Register des VIC-II-Chip (siehe letzte Folge). Dort ist Bit 7 gesetzt, wenn die Anforderung vom VIC-II-Chip kam.
602B	**STA D019**	Zurückschreiben
602E	**BMI 6037**	Sprung, falls VIC-IRQ, sonst CIA-IRQ.
Bearbeiten eines CIA-IRQ:		
6030	**LDA DC0D**	Löschen des CIA1 Unterbrechungs-Kontrollregisters.
6033	**CLI**	IRQ zulassen. Damit können innerhalb eines CIA- IRQ auch unsere VIC-IRQs geschehen.
6034	**JMP EA31**	Bearbeitung des CIA-IRQ durch die normale Routine.

Unser Programm für VIC-II-IRQs:

6037	**LDA D012**	Rasterzeilen-Register laden um festzustellen, welche Zeile den IRQ auslöste.
603A	**CMP #$F8**	Vergleich mit Ende des Textfensters.
603C	**BCS 604F**	Wenn unterhalb des Textfensters, Sprung.

Der folgende Programmteil ist wirksam, wenn der IRQ-Auslöser eine Zeile in Höhe des Textfensters war:

603E	**CLC**	Addition vorbereiten.
603F	**ADC 02**	Streifenbreite aus dem Merkregister addieren.
6041	**STA D012**	Neuen Wert in Rasterzeilen-Register schreiben.

Damit wird eine neue Rasterzeile als IRQ-Auslöser festgelegt, die um die Streifenbreite tiefer liegt als die vorhergegangene.

Es folgt eine kleine Verzögerungsschleife, die aber nur zum Experimentieren eingebaut wurde:

6044	**LDY #$03**	Schleifen-Startwert
6046	**DEY**	Herunterzählen
6047	**BNE 6046**	NEXT Y, bis Y=0.

Ändern der Rahmenfarbe bis zum nächsten Raster-IRQ:

6049	**INC D020**	Farbcode+1. Wenn Code im Rahmenfarbregister größer als 15 wird, fängt wieder Farbcode 0 an, weil die Bits 5-7 keine Funktion haben.

Abschließend erfolgt der Rücksprung in den Rest der normalen IRQ-Routine:

604C	**JMP EA81**	Siehe unsere Untersuchung der IRQ-Firmware.

Damit ist der Rahmen in Höhe des Textfensters behandelt. Es schließt sich nun der Teil an, der die Rahmenbereiche unter- und oberhalb bearbeitet:

604F	**LDA #$00**	Farbcode schwarz
6051	**STA D020**	in Rahmenfarb-Register.
6054	**LDA #$32**	Rasterzeile, bei der oben das Textfenster beginnt.
6056	**STA D012**	In Rasterzeilen-Register schreiben
6059	**JMP EA81**	Abschluß durch Sprung zum Ende der normalen IRQ- Routine.

Damit ist festgelegt, daß ober- und unterhalb des Textfensters die Rahmenfarbe schwarz wird.

Unsere eigene Routine ist jetzt abgeschlossen. Zum guten Ton gehört es, dem Benutzer auch die Möglichkeit zu öffnen, diese Routine wieder abzuschalten. Das erfolgt im letzten Programmteil, der mittels SYS24688 aktiviert werden kann:

605C	**SEI**	IRQ sperren
605D	**LDA #$00**	Raster-IRQ
605F	**STA D01A**	abschalten
6062	**LDA #$31**	IRQ-Vektor
6064	**STA 0314**	restaurieren

Fortsetzung auf Seite 131

```
PROGRAMM 2

,6000  78        SEI
,6001  A9 28     LDA #28
,6003  8D 14 03  STA 0314
,6006  A9 60     LDA #60
,6008  8D 15 03  STA 0315
,600B  A9 F8     LDA #F8
,600D  8D 12 D0  STA D012
,6010  AD 11 D0  LDA D011
,6013  29 7F     AND #7F
,6015  8D 11 D0  STA D011
,6018  A9 81     LDA #81
,601A  8D 1A D0  STA D01A
,601D  A9 00     LDA #00
,601F  8D 20 D0  STA D020
,6022  A9 04     LDA #04
,6024  85 02     STA   02
,6026  58        CLI
,6027  60        RTS
-------------------------
,6028  AD 19 D0  LDA D019
,602B  8D 19 D0  STA D019
,602E  30 07     BMI 6037
,6030  AD 0D DC  LDA DC0D
,6033  58        CLI
,6034  4C 31 EA  JMP EA31
-------------------------
,6037  AD 12 D0  LDA D012
,603A  C9 F8     CMP #F8
,603C  B0 11     BCS 604F
,603E  18        CLC
,603F  65 02     ADC   02
,6041  8D 12 D0  STA D012
,6044  A0 03     LDY #03
,6046  88        DEY
,6047  D0 FD     BNE 6046
,6049  EE 20 D0  INC D020
,604C  4C 81 EA  JMP EA81
-------------------------
,604F  A9 00     LDA #00
,6051  8D 20 D0  STA D020
,6054  A9 32     LDA #32
,6056  8D 12 D0  STA D012
,6059  4C 81 EA  JMP EA81
-------------------------
,605C  78        SEI
,605D  A9 00     LDA #00
,605F  8D 1A D0  STA D01A
,6062  A9 31     LDA #31
,6064  8D 14 03  STA 0314
,6067  A9 EA     LDA #EA
,6069  8D 15 03  STA 0315
,606C  A9 0E     LDA #0E
,606E  8D 20 D0  STA D020
,6071  58        CLI
,6072  60        RTS
-------------------------
```

Listing 2. Das im Artikel entwickelte Programm auf einen Blick

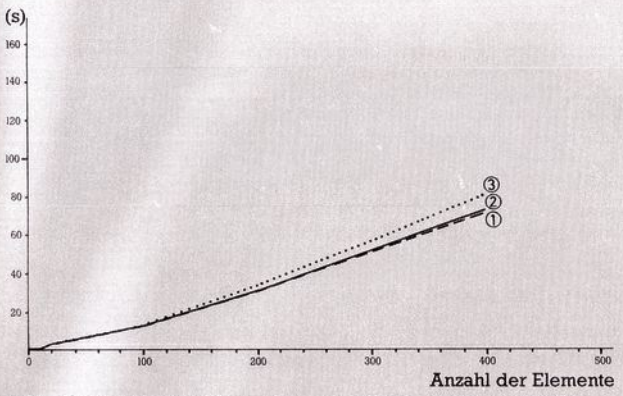

Bild 9. Heapsort zählt zu den schnellsten Sortiermethoden, wie die flache Kurve zeigt

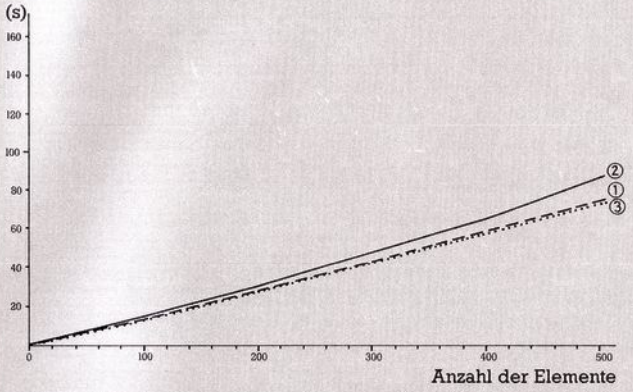

Bild 10. Quicksort war lange der schnellste Sortieralgorithmus

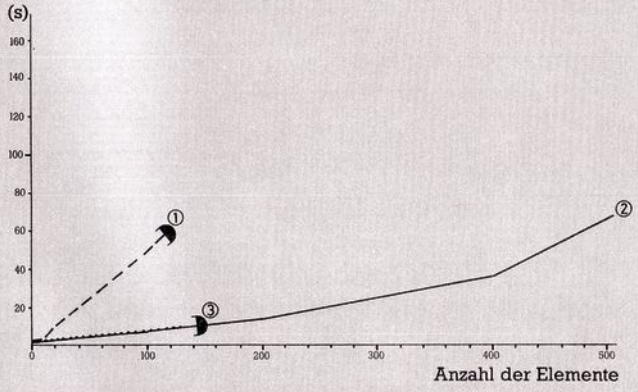

Bild 11. Supersort zeigt einige Mängel auf dem C 64, kann aber spezialisiert hervorragend eingesetzt werden

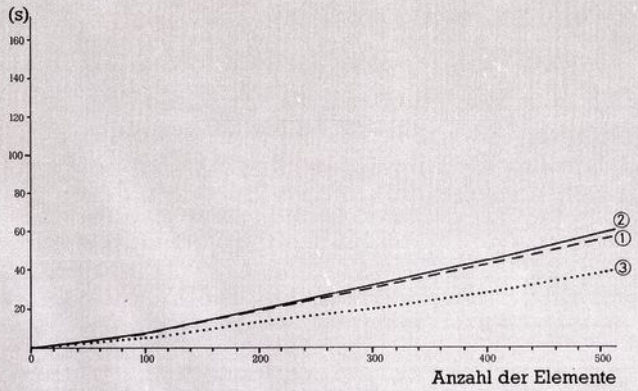

Bild 12. Mischsort besticht durch seine Schnelligkeit in allen Bereichen

lection. Supersort benötigt nämlich mehr Zwischenspeicher als Quicksort und wird somit von diesem überholt, sobald bei Supersort die Garbage Collection in Aktion tritt (und das tut sie natürlich früher als bei Quicksort).

Natürlich kann man jetzt den Entwickler dieses Programms nicht des Lügens bezichtigen, wenn er behauptet: »Supersort ist schneller als Quicksort!«. Supersort ist nämlich ursprünglich auf einem CBM 8032 geschrieben worden, und dieser verfügt über eine andere Stringverwaltung als der C 64, was eine erheblich schnellere Garbage Collection zur Folge hat

Für den C 64 gilt jedoch (leider): Supersort ist nur bedingt brauchbar, da es einige Schwächen aufweist, die Quicksort nicht hat.

Nun zu Bild 12. Es zeigt den letzten der Sortieralgorithmen, nämlich Mischsort. Und hier erlebte ich die große Überraschung: Auch Mischsort ist schneller als die Normalversion von Quicksort (ohne Tricks mit kleineren Sortierroutinen). Mischsort zeigte im Test wahrhaftig traumhafte Zeiten für Basic-Sortierprogramme, die teilweise nur 50 Prozent der Zeiten von Quicksort betrugen. Es wurde auch deutlich, daß bei Mischsort die Garbage Collection noch später einsetzte als bei Quicksort (das Einsetzen der Garbage Collection ist an den »Knickstellen« der Graphen zu erkennen), was die guten Zeiten aber nur teilweise rechtfertigte.

Mischsort ist genauso einsatzfähig wie Quicksort (ohne »Macken«) und kann deshalb als bestes Sortierprogramm dieses Tests betrachtet werden. Haben Sie viele große Felder zu sortieren, so spielt es bei diesem Algorithmus fast keine Rolle, ob die Felder vorsortiert sind oder nicht; schnell ist er auf jeden Fall.

Noch eine Randbemerkung zum Schluß: Vielleicht haben Sie auch schon Methoden entwickelt, wie sie die Garbage Collection »in den Griff bekommen«. Haben Sie vielleicht als »Maschinensprachefreak« ein neues Basic-ROM geschrieben (mit besserer Stringverwaltung) oder haben Sie anhand unseres Vorkurses über Strings die Lösung des Problems gefunden?

Die 64'er-Redaktion würde sich über Beiträge aus dem Leserkreis freuen. Vielleicht möchten Sie dazu beitragen, daß auch andere Anwender in den Genuß entsprechender Verbesserungen kommen?

Schreiben Sie uns doch! Bestimmt können wir dann den einen oder anderen Trick mit in unsere Reihe aufnehmen.

Bis zum nächsten Mal.

(K.Schramm/gk)

```
6067    LDA #$EA
6069    STA 0315
606C    LDA #$0E
606E    STA D020
6071    CLI
6072    RTS
```

Unser Programm ist komplett. Speichern Sie es bitte vor dem Starten ab. Nach dem SYS 24576 finden Sie einen hübschen bunten Rahmen vor, oberhalb und unterhalb des Textfensters ist er schwarz. Besonders gut — finde ich — sieht das Ganze aus, wenn man die Hintergrundfarbe des Textfensters auch auf Schwarz setzt. Das Programm erlaubt noch einige Experimente:

Durch POKE-Kommandos in die Speicherstelle 2 kann die aktuelle Streifenbreite variiert werden, durch POKEs in die Zelle 24645 der Startwert der Verzögerungsschleife. Probieren Sie's doch mal aus. Eine Erkenntnis werden Sie gewinnen: In der Unterbrechungs-Programmierung spielt die Zeit eine wichtige Rolle. Das zeigt sich auch, wenn man zum Beispiel Cursorbewegungen durchführt: Die Streifen fangen an zu wandern.

Weitere Möglichkeiten zum Experimentieren sind gegeben, wenn Sie die Rasterzeilen verändern, die den oberen und unteren Rand des Textfensters markieren:

Durch POKE 24661,Zahl verschieben Sie die obere, durch POKE 24635,X:POKE 24588,X die untere Rasterzeile, von der an alles schwarz ist. Wie schon vorhin erwähnt, habe ich im Programm diese Werte auf 50 beziehungsweise 248 fixiert, weil genau dort auf meinem Monitor das Textfenster liegt.

Mit diesem Beispiel und dem aus der Grafikserie sollte es Ihnen nun möglich sein, auch andere Unterbrechungsprogramme zu schreiben, die sich der Rasterzeilen-Unterbrechung per VIC-II-Chip bedienen. Eine Bemerkung sollte ich Ihnen noch auf den Weg Ihrer eigenen Versuche mitgeben: Der Elektronenstrahl, der über den Bildschirm saust und beim Erreichen des von uns bestimmten Rasterzeilenwertes zum Auslösen des IRQ führt, ist enorm schnell. Die Serviceprogramme dürfen deshalb nicht zu lang sein, sonst steht der nächste IRQ schon wieder an, bevor der vorangegangene bearbeitet ist.

In der kommenden Folge sollen Beispiele für andere Unterbrechungsformen vorgestellt werden, die durch die CIAs und die RESTORE-Taste angesprochen werden. Danach soll es um die Anwendung des bisher Gelernten gehen, wobei wir uns wieder mehr den Interpreter-Routinen und auch den Kernal-Möglichkeiten zuwenden wollen.

(Heimo Ponnath/gk)