Hires-3 (Teil 3)

Text und Grafik mischen ist ein vielschichtiges Thema. Wir geben Ihnen eine ausführliche Anleitung zur Hand, mit der Sie dieses Problem relativ einfach bewältigen können.

Zwei Varianten sind denkbar, Text und Grafik zu mischen.
1.) Wir mischen auf dem Bildschirm zwei verschiedene Darstellungsmodi, nämlich den Text- und den Hochauflösungsmodus. Das ist per Rasterzeileninterrupt zu erreichen. Ein Beispiel fanden Sie in der siebten Folge des Grafikkurses (64'er, Ausgabe 10/84).
2.) Wir sorgen dafür, daß die Schrift in die Bit-Map eingetragen wird. Auch dafür sehen Sie in der siebten Folge ein einfaches Beispiel in Basic.

Wir wollen uns mal die Vor- und Nachteile der beiden Möglichkeiten vor Augen halten: Die Lösung mittels Rasterzeileninterrupt bietet den Vorzug, daß die Bildschirmaufspaltung ständig vorhanden sein kann, wenn sie einmal angeschaltet wurde. Außerdem kann man alle Textausgaben bei geeigneter Cursorsteuerung lesbar halten, sogar Fehlermeldungen oder andere unerwartete Texte. Man kann im Direktmodus trotz vorhandenem Grafikbild lesbare Eingaben machen, oder im Programm-Modus INPUT-Abfragen etc. erlauben. Als Nachteile stehen dem gegenüber: Grafik und Schrift müssen unter- oder übereinander angeordnet werden. Der Rasterzeileninterrupt ist nämlich nur zeilenweise schaltbar. Es ist also beispielsweise nicht möglich, die linke Bildschirmhälfte im Hochauflösungsmodus und die rechte im Textmodus zu verwenden. Ein weiteres Manko ist es, daß unter gewissen Umständen ein Flimmern des Bildschirms auftreten kann. Durch sorgfältiges Programmieren der Interruptroutine ist eine Quelle dafür zwar zu beseitigen, aber Probleme treten auf, wenn das Betriebssystem ein Hochscrollen des Textes erzwingt. Doch dazu später. Ein letzter Nachteil ist, daß man sehr auf andere Interruptroutinen — besonders solche, die unseren Computer funktionsfähig halten, achten muß. Aber das ist programmtechnisch lösbar.

Nun zur zweiten Variante: Da ist zunächst mal der unbestreitbare Vorzug, daß der Text an jeder beliebigen Stelle auftreten kann, ja sogar mitten in der Grafik, denn er ist ja jetzt selbst Grafik. Außerdem könnte man den Text noch vergrößern oder sonstwie anders gestalten. Das letztere — gleich hier soll's gesagt sein — ist aber in dieser Folge nicht eingeschlossen. Als Nachteile stehen dem gegenüber, daß der Text vorher definiert werden muß, daß also keine spontanen Regungen unseres Computers — wie Fehlermeldungen — damit erfaßbar sind. Außerdem sind wir es gewohnt, daß für diese Art der Test-in-Grafik-Programmierung das Zeichen-ROM in einen zugreifbaren Speicherbereich kopiert werden muß (wie zum Beispiel in Folge 7). Das aber verschlingt eine Unmenge an Speicherplatz. Weil es eines der Ziele von HIRES-3 ist, keinen unnötigen RAM-Speicher zu verschleudern, werden wir uns einer programmtechnischen Lösung dieses Problems bedienen.

Als Fazit ergibt sich, daß wir möglichst beide Versionen in Hires-3 einbauen sollten, um sowohl Fehlermeldungen und Direkteingaben als auch Text in der Grafik zu ermöglichen. Das soll Schritt für Schritt in dieser Folge geschehen. Die Programmsprache, die wir einsetzen, wird Assembler sein, und wenn Sie den Kurs zur Maschinensprache »Assembler ist keine Alchimie« lesen, dann haben Sie hier die Gelegenheit, einige Anwendungen zu erarbeiten und eventuell nach eigenen Bedürfnissen umzubauen, denn: Es gibt kein Programm, an dem nicht noch etwas zu verbessern wäre.

Rasterzeileninterrupt

Zunächst einmal, »interrupt« heißt auf deutsch »Unterbrechen«. Unser Computer — von uns unbemerkt — unterbricht viele Male pro Sekunde das, woran er gerade arbeitet, um wichtige Parameter aufzufrischen. Es gibt mehrere Sorten dieser Interrrupts, uns soll hier nur der interessieren, den wir nutzen wollen: der sogenannte IRQ. Dieser Interrupt kann softwaremäßig gestattet oder unterdrückt werden durch zwei Assembler-Befehle (SEI = setze IRQ-Flagge = Verhindern von IRQ, CLI = lösche IRQ-Flagge = Erlauben von IRQ). Außerdem kann in einigen Registern noch bestimmt werden, welche Ereignisse einen IRQ auslösen dürfen.

Sehen wir uns zuerst nochmal an, welchen Weg ein unbeeinflußter IRQ nimmt. Ganz hinten in unserem Speicher (65534/65535) steht eine Adresse (65352), die beim sogenannten Hardware-Interrupt angesprungen wird. An dieser Zieladresse 65352 werden zunächst alle Register an einen sicheren Platz gerettet, schließlich aber mittels eines indirekten Sprunges die eigentliche IRQ-Routine angesteuert. Der indirekte Sprung erfolgt zu der Adresse, die in den Speicherstellen 788/789 ($314/315) enthalten ist. Das sind RAM-Zellen, die also von uns verändert werden können. Behalten wird diese Tatsache erst mal im Gedächtnis und sehen uns den normalen weiteren Verlauf an. Normalerweise ist in diesem IRQ-Vektor als Zieladresse $EA31 (dezimal 59953) enthalten. Die an dieser Stelle startende IRQ-Routine wird im Normalfall alle 1/60 Sekunden aufgerufen. Darin wird die interne Uhr weitergestellt, der Cursor geschaltet, Ein- und Ausgabe-Parameter abgefragt, die Tastatur auf Eingaben beobachtet, etc. Abschließend holt das Programm wieder die zu Beginn geretteten Register zurück und schaltet zur normalen Tätigkeit des Computers weiter. Das Interruptprogramm ist dann bis zur nächsten 1/60 Sekunde beiseite gelegt. Diese normale Routine wird durch die Timer der CIA-Bausteine unseres Computers gesteuert.

Der übliche Weg, den auch wir beschreiten werden, ist, den Vektor 788/789 auf eine selbst programmierte IRQ-Routine zu stellen und diese dann mit einem Sprung in das Ende der normalen IRQ-Routine abzuschließen. Von dem Moment an durchläuft alle 1/60 Sekunden der Computer unsere eigene Routine.

Wie muß diese Routine aussehen? Unser Ziel soll es sein, daß eine Text-Kopfzeile auf dem Bildschirm sichtbar ist und daß die unteren vier Zeilen ebenfalls im Textmodus erscheinen. Dazwischen soll der Hochauflösungsmodus dargestellt werden (siehe Bild 1).

Das sind Aufgaben, die der VIC-II-Chip zu erledigen hat. Dafür ist er ebenfalls mit einer IRQ-Steuermöglichkeit ausgestattet. Zwei Register spielen hier die entscheidende Rolle:

53273 ($D019)	Interrupt Latch Register, auch Interrupt Request- oder Interrupt Status-Register genannt.
53274 ($D01A)	Interrupt Enable Register

Der Aufbau beider Speicherstellen ist identisch. Bit 0 ist die zum Rasterinterrupt gehörige Flagge, Bit 1 hat mit Sprite/Hintergrund-Kollisionen zu tun, Bit 2 mit Kollisionen von Sprites untereinander, Bit 3 wird bei der Lichtgriffel-Benutzung angesprochen. Die Bits 4, 5 und 6 sind unbenutzt. Bit 7 ist immer dann gesetzt (oder muß in 53274 gesetzt werden), wenn eines der anderen Bits angesprochen wird (siehe Bild 2).

Der Unterschied beider Register ist der, daß 53273 lediglich **anzeigt**, daß eine der vier möglichen Interrupt-Quellen einen IRQ ausgelöst hat. In dem Fall ist Bit 7 gesetzt, und das Bit des auslösenden Ereignisses ist gleich 1. Wir kennen sowas noch von der Folge 5, wo es um Kollisionen von Sprites ging. Bei einem Rasterzeileninterrupt findet man dann Bit 7 und Bit 0 gesetzt. Welcher Interrupt von den vier möglichen überhaupt zugelassen wird, kann man im Register 54274 bestimmen. Bit 7 regelt, ob überhaupt einer erlaubt wird (von den vier erwähnten). Ist Bit 7 gesetzt, sind solche IRQs gestattet. Durch Setzen der Bits 0 bis 3 wird die auslösende Quelle festgelegt. Dabei sind auch mehrere möglich. Man nennt das dabei gebildete Bit-Muster die Interrupt-Maske. Wenn wir nur den Rasterinterrupt zulassen wollen, müssen wir also Bit 0 und Bit 7 auf 1 setzen.

Rasterzeileninterrupt bedeutet, daß ab einer bestimmten Rasterzeile unser Computer in das Interruptprogramm springen soll, welches wir durch Einschreiben in den Vektor 788/789 definiert haben. Dazu muß dem VIC-II-Chip natürlich noch gesagt werden, welche Rasterzeile wir wählen wollen. Falls Sie über den Begriff »Rasterzeile« stolpern, dann sehen Sie in der 8. und 7. Folge der Grafik-Serie nochmal nach, wie der Computer das Bild auf Ihrem Monitor (oder Fernsehgerät) zusammenbaut. Diese Mitteilung an den VIC-II-Chip geschieht wieder über zwei Register:

53265 ($D011)	Hiervon aber nur Bit 7
53266 ($D012)	Das ganze Register

Die Sache verhält sich wie bei der X-Position von Sprites: Es können Zahlen auftreten, die größer als 255 sind. Wir haben den ganzen Bildschirm in 280 Rasterzeilen vorliegen (wobei das sichtbare Fenster etwa von Zeile 40 bis Zeile 240 reicht). Um beispielsweise die größtmögliche Rasterzeilen-Zahl 280 binär darzustellen, braucht man 9 Bits:

1 0001 1000

Dieses neunte Bit schreibt man als Bit 7 ins Register 53265, die restlichen acht Bit bilden den Inhalt des Registers 53266.

Wir planen ja die erste Zeile im Text- und den weiteren Bildschirminhalt bis zur viertletzten Zeile im Hochauflösungsmodus. Durch ein bißchen Probieren bekommt man heraus, daß der Moduswechsel einmal in der 58. Rasterzeile und dann wieder in der 218. Rasterzeile stattfinden muß. Von da an kann der Bildschirm weiter im Textmodus

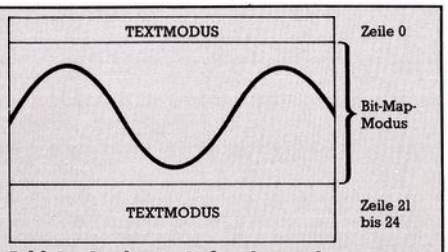

Bild 1. Geplante Aufspaltung des Bildschirmes per Rasterzeilen-Interrupt

Bits	7	6	5	4	3	2	1	0
IRQ-Flagge für	eine der 4 IRQ-Quellen ist angesprochen	UNBENUTZT			Lichtgriffel	Sprite-Sprite-Kollision	Sprite-Hintergr.-Kollision	Rasterzeilen-IRQ

Bild 2. Aufbau der Register 53273 und 53274

bleiben, bis nach dem Null-Übergang wieder Rasterzeile 58 gefunden wird. Obwohl wir letztlich den Bildschirm in drei Teile auftrennen (1. Zeile Text, dann Grafik, 21. bis 25. Zeile wieder Text) brauchen wir nur zwei Moduswechsel (Rasterzeile 58 bis 217 Grafik, Rasterzeile 218 — 57 Text). Sowohl 58 als auch 218 sind noch in acht Bit darzustellen, Bit 7 aus Register 53265 bleibt somit unverändert Null.

Jetzt wissen wir alles, was wir zur Anwendung des Rasterzeilen-Interrupt brauchen, außer... dem eigenen Interruptprogramm. Das soll nun vorgestellt werden. Zuvor aber noch eine Bemerkung an diejenigen, die (noch!) keine Assemblerprogrammierung betreiben. Die Maschinensprachroutine wird von mir ausführlich erklärt, weil man nur sehr wenig Literatur zu diesem Thema findet. Sollten Sie die Routine nutzen wollen, ohne genau wissen zu wollen, wie es programmtechnisch gemacht werden kann, dann geben Sie sie einfach nach dem beigefügten Listing (Programm 1) mittels MSE ein.

Das gesamte Programm besteht aus drei Teilen: Anschalten (Initialisieren) des Rasterzeileninterrupt, eigentliche Interrupt-Routine und Abschalten. Bei der Initialisierung muß zunächst der Inhalt der Textfenster gelöscht werden, denn dort steht ja für den Hochauflösungsmodus der Farbcode drin. Das geschieht in zwei Schleifen:

```
89B8  LDA #20       Code für »Space« in Akku
89BA  LDX #27       X-Register als Index mit dezimal 39 geladen
89BC  STA 8C00,X    Leerzeichen in Zeile 0 des Bildschirmspeichers beginnt in HIRES-3 ja bei 8C00)
89BF  STA 8F48,X    und in die 21. Zeile
89C2  DEX
89C3  BPL 89BC      das geschieht so lange, bis 40 Leerzeichen eingeschrieben sind, also die Zeilen 0 und 21 gelöscht wurden.
```

Jetzt kümmern wir uns noch um die letzten drei Zeilen:

```
89C5  LDX #77       X-Indez auf 119
89C7  STA 8F70,X    das Leerzeichen wird nun in die letzten drei Zeilen geschrieben
89CA  DEX           bis alle 120 Bildschirmpositionen gelöscht
89CB  BPL 89C7      sind.
```

Nun kommen wir zum Verbiegen des IRQ Zeigers:

```
89CD  SEI           Während dieser Prozedur können wir keine Interrupts gestatten
89CE  LDA #EC       LSB der Startadresse unserer IRQ-Routine
89D0  STA 0314      in LSB des IRQ-Zeigers.
89D3  LDA #89       MSB der Startadresse
89D5  STA 0315      in MSB des IRQ-Zeigers.
```

Wir schreiben nun unsere erste Rasterzeile (58= $3A) in das Rasterzeilenregister 53265/53266:

```
89D8  LDA #3A       Nummer der Rasterzeile, von der an vom Text- in den Grafik-Modus umgeschaltet wird. Im weiteren obere Position genannt.
89DA  STA D012      das ist Register 53266
89DD  LAD D011      Register 53265 wird in den Akku geladen
89E0  AND #7F       mit der AND-Maske $7F = binär 0111 1111 wird ein eventuell vorhandenes Bit 7 gelöscht
89E2  STA D011      der so veränderte Inhalt wird ins Register zurückgeschrieben.
```

Als letztes in der Initialisierungsphase müssen wir noch eine Maske ins Interrupt Enable Register 53274 schreiben um anzuzeigen, daß und vor allem welchen IRQ wir zulassen:

```
89E5  LAD #81       das ist binär 1000 0001
89E7  STA D01A      das ist das IRQ Enable Register
89EA  CLI           jetzt dürfen wieder Interrupts geschehen
89EB  RTS           Rücksprung zum aufrufenden Programm.
```

Von nun an durchläuft jeder IRQ-Aufruf unsere ab $89EC vorhandene Routine. Das betrifft sowohl die IRQs, die von den Timern des CIA stammen, als auch die Rasterzeileninterrupts. Dann wollen wir mal schleunigst dafür sorgen, daß dort auch wirklich eine Routine steht! Zuerst überprüfen wir, ob eine Interruptanforderung auch wirklich von VIC-II-Chip her kommt:

```
89EC  LDA D019      Wir laden das Interrupt Request Register 53273 in den Akku
89EF  STA D019      und löschen es sofort wieder durch zurückschreiben
89F2  BMI 89FB      war Bit 7 gesetzt, dann lag ein IRQ vom VIC-II-Chip vor, also unser Rasterzeileninterrupt. In diesem Fall überspringen wir die nächsten Zeilen.
```

War Bit 7 in diesem Register nicht gesetzt, dann kam die IRQ-Anforderung vom CIA und wir benutzen die normale IRQ-Routine:

```
89F4  LDA DC0D      das ist das IRQ-Register des CIA und wir müssen den Anfang der normalen IRQ-Routine simulieren. Das geschieht hier durch Auslesen des CIA-Register (hier wird es auf diese Weise gelöscht)
89F7  CLI           wir löschen die IRQ-Flagge, um während des Timer-Interrupt einen Rasterzeileninterrupt zuzulassen
89F8  JMP EA31      wir springen zur normalen IRQ-Routine.
```

Nun kommt der Teil, den wir per Rasterzeileninterrupt ansteuern. Erst mal müssen wir feststellen, ob die IRQ-Anforderung durch die obere oder die untere Position erfolgt ist:

```
89FB  LDA D012      das ist das Rasterzeilenregister 53266
89FE  CMP #DA       $DA = dezimal 218, also die untere Position
8A00  BCS 8A1F      kam die IRQ-Anforderung durch die untere Position zustande, dann wird zur dazugehörigen Routine verzweigt.
```

Nach all diesen Vorkehrungen kommt der Programmablauf hier an, wenn die obere Position für einen Rasterzeileninterrupt verantwortlich ist. Hier soll der Wechsel vom Text- zum Hochauflösungsmodus stattfinden. Für das Anschalten dieses Modus waren ja (siehe Folge 3 der Grafik-Serie) die Register 53265 ($D011) und 53272 ($D018) zuständig:

```
8A02  LDA #38       Maske binär 0011 1000 in Akku
8A04  LDY #3B       Make binär 0011 1011 in Y-Register
8A06  STA D018      Akku-Maske in Register 53272
8A09  STY D011      Y-Register-Maske in Register 53265.
```

Damit wurde der Hochauflösungsmodus eingeschaltet und im folgenden Bildschirmteil wird der Bit-Map-Inhalt dargestellt. Es gibt nun ein Problem, das ich in den nächsten Programmzeilen einigermaßen lösen möchte. Unter dem Grafikbild werden wieder 4 Textzeilen eingerichtet. Sobald der Text dort über Zeile 24 hinausreicht, erzwingt das Betriebssystem ein Hochscrollen des Bildschirm-RAM-Inhaltes. Das drückt sich an zwei Stellen aus: Textzeilen schieben sich in den unteren Teil des Grafik-Bildes hinein, wo sie als farbige Quadrate stören. Zum zweiten scrollt der Inhalt der Kopfzeile aus dem Bildschirm und dafür treten die Farbcodes aus dem oberen Teil des Grafik-Bildes dort hinein und zeigen ein Sammelsurium von Zeichen. Für das zweite Problem, also die Zerstörung der Kopfzeile, werde ich hier keine Lösung angeben. Die finden Leser des Assembler-Kurses in der Ausgabe 2/1985 des 64'er-Magazins. Mit ein wenig Geschick können Sie das Programm zum Rückschreiben der Kopfzeile um- und hier einbauen. Aber auch das andere Problem ist zwar gelöst, aber noch nicht ganz zufriedenstellend. Wir schreiben einfach bei jedem Rasterzeilen-IRQ in den unteren Rand des Grafik-Schirmes die Farbcodes hinein:

```
8A0C  LDX #27       das ist wieder der Zähler, den wir schon von der Initialisierung her kennen
8A0E  LDA 8E26      aus irgendeinem Bildschirmspeicherplatz des Hochauflösungsbildes wird der Farbcode entnommen und in den Akku gelegt
8A11  STA 8F20,X    dieser Farbcode wird in die letzte Grafikzeile geschrieben
8A14  DEX
8A15  BPL 8A11      das geschieht, bis die ganze Zeile neu beschrieben wurde.
```

Es zeigt sich, daß auf diese Weise das Problem zwar gelöst wurde, daß sich aber bei häufigem Scrollen, zum Beispiel beim LISTen eines Programmes, manchmal ein kleines Flackern ergibt. Eine andere Möglichkeit, diese Scroll-Frage in den Griff zu bekommen, wäre natürlich die Veränderung der Scroll-Routine des Betriebssystems gewesen. Dazu hätte man allerdings die RAM-Bereiche unter den ROMs verwenden müssen, was — abgesehen von einer Unmenge verplemperten Speicherplatzes — auch Schwierigkeiten mit unserer Bit-Map unter dem Basic-ROM ergeben hätte. Falls Sie eine bessere Lösung wissen, dann schreiben Sie mir. So können wir vielleicht gemeinsam Hires-3 vervollkommnen.

Aber unser Programm ist noch nicht fertig. Wir müssen an den zweiten Moduswechsel denken. Dazu schreiben wir in das Rasterzeilenregister jetzt die untere Position ein:

```
8A17  LDA #DA     das ist die Rasterzeilen-Nummer der unteren
                  Position
8A19  STA D012    da haben wir wieder unser Rasterzeilen-
                  Register. Von nun an wird der IRQ von die-
                  ser unteren Position ausgelöst.
8A1C  JMP EA81    schließlich springen wir zum Ende der nor-
                  malen IRQ-Routine.
```

Wenn jetzt der nächste Rasterzeilen-Interrupt ausgelöst wird, dann muß er auf ein Programm laufen, das den Textmodus einrichtet:

```
8A1F  LDA #34     Maske binär 0011 0100 in Akku
8A21  LDY #1B     Maske binär 0001 1011 in Y-Register
8A23  STA D018    Akku-Maske in Register 53272
8A26  STA D011    Y-Register-Maske in Register 53265
```

Damit ist der Textmodus wieder eingeschaltet. Wir müssen nun noch dafür sorgen, daß der Wert der oberen Position ins Rasterzeilenregister eingetragen wird:

```
8A29  LDA #3A     dies ist unsere obere Position
8A2B  STA D012    Wir stellen das Rasterzeilenregister wieder
                  auf diese obere Position
8A2E  JMP EA81    Abschließend erfolgt wieder der Sprung
                  zum Ende der normalen IRQ-Routine.
```

Damit ist die eigentliche IRQ-Routine abgeschlossen. Wenn wir aber jemals wieder zu normalen Verhältnissen zurückkehren wollen, dann sollten wir auch noch einen Programmteil zum Abschalten des Rasterzeileninterrupt konstruieren. Das geschieht zunächst einmal durch Löschen des Interrupt Request Registers im VIC-II-Chip:

```
8A31  SEI         Wir wollen beim Abschalten nicht durch um-
                  hervagabundierende IRQs gestört werden.
8A32  LDA #00
8A34  STA D01A    Durch Einschreiben einer Null wird Register
                  53274 gelöscht
```

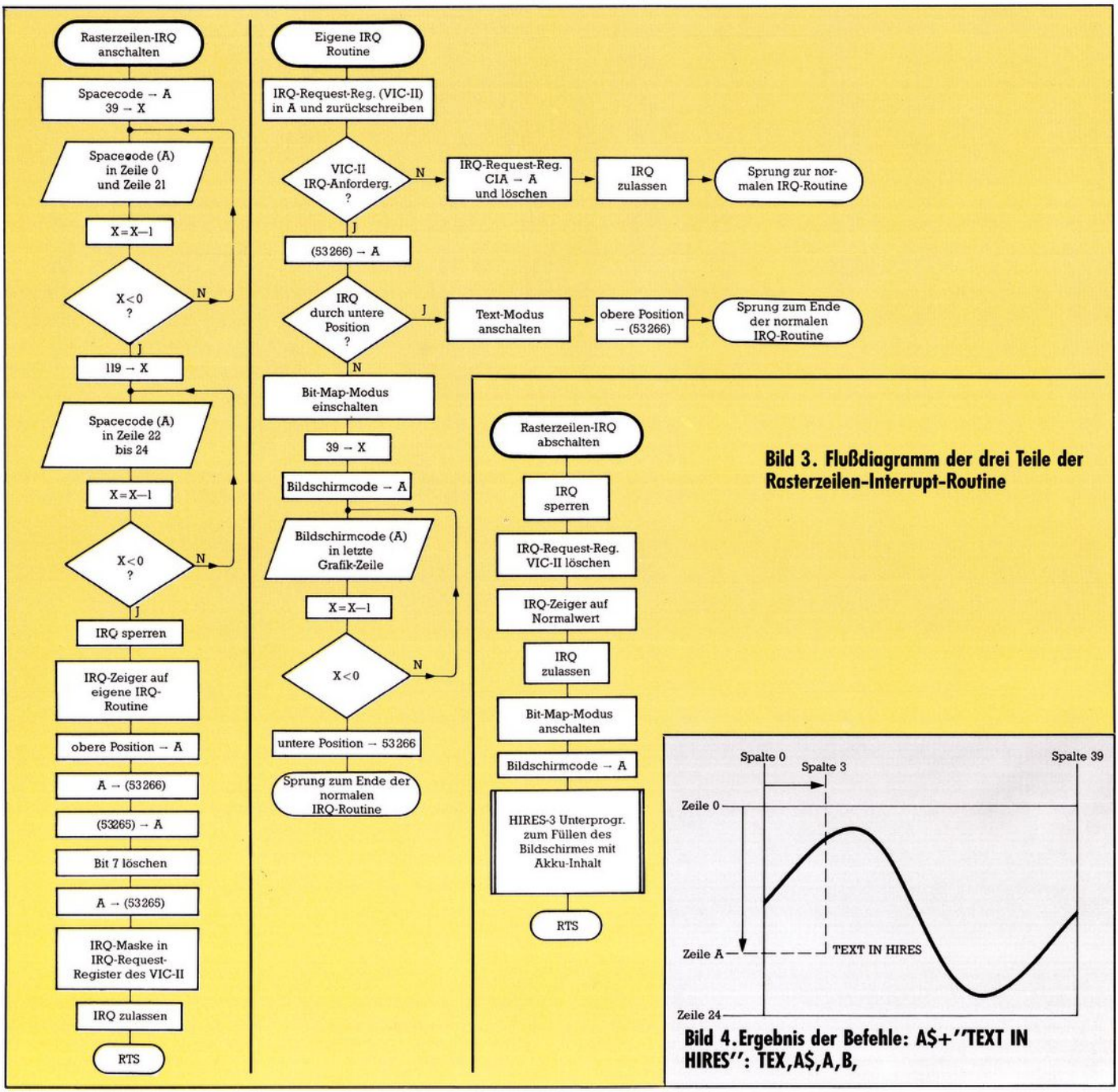

Bild 3. Flußdiagramm der drei Teile der Rasterzeilen-Interrupt-Routine

Bild 4. Ergebnis der Befehle: A$+"TEXT IN HIRES": TEX,A$,A,B,

Dann stellen wir den IRQ-Zeiger wieder auf die normale Routine $EA31:

```
8A37  LDA #31     LSB der IRQ-Adresse
8A39  STA 0314    in LSB des IRQ-Zeigers
8A3C  LDA #EA     MSB der IRQ-Adresse
8A3E  STA 0315    in MSB des IRQ-Zeigers
8A41  CLI         Jetzt darf wieder unterbrochen werden
```

Zu guter Letzt soll nach Beendigung des Rasterzeilen-Interrupt der Hochauflösungszustand wiederhergestellt werden für den ganzen Bildschirm, und die Eintragungen in den Textzeilen müssen gegen den Farbcode ausgetauscht werden:

```
8A42  LDA #38
8A44  LDY #3B
8A46  STA D018
8A49  STY D011    Das alles kennen Sie schon von weiter oben
                  im Programm (ab 8A02)
8A4C  LDA 8E26    wieder laden wir den Farbcode einer beliebigen Bildschirmspeicherzelle in den Akku
8A4F  JSR 9532    Diese Routine füllt den gesamten Bildschirmspeicher mit dem Akku-Inhalt.
8A52  RTS         Rücksprung zum aufrufenden Programm
```

Das war's! Mit SYS 35256 schalten Sie die Bildschirm-Aufspaltung ein, mit SYS 35377 wieder aus. Ein Flußdiagramm dieser Routine finden Sie in Bild 3.

Als Programm 1 ist diese Routine zur Eingabe mittels MSE angefügt. Programm 3 ist ein Basic-Aufrufprogramm, das aber außer dieser neuen Implementierung noch die zweite Version beansprucht. Bevor Sie Programm 3 also starten, bauen Sie zunächst noch Programm 2, nämlich das direkte Einschreiben von Text in die Bit-Map in Hires-3 ein. Wie man alles zusammensetzt, wird Ihnen am Ende dieser Folge noch erklärt werden.

Zeichen in die Bit-Map schreiben

Zunächst legen wir fest, auf welche Weise wir die Eingaben machen wollen. Der darzustellende Text soll sowohl als Stringvariable (zum Beispiel B$), als auch als direkter String (zum Beispiel »TEST«) als auch als Stringfeldvariable (zum Beispiel B$(1)) und schließlich auch noch als Stringfunktion (zum Beispiel MID$(B$,3,2)+STR$(A)) anzugeben sein. Alle in Basic erlaubten String-Erscheinungsformen dürfen also auftreten. Weiterhin sollen Zeile und Spalte das Stringanfangs anzugeben sein. Das ganze wird schließlich noch mit einem neuen Befehlswort »TEX« in Hires-3 eingebaut (Bild 4). Doch dazu später. Die Syntax soll dann lauten: **TEX, String, Zeile, Spalte**

Die Angaben Zeile, Spalte dürfen ebenfalls in jeder erdenklichen, in Basic erlaubten Form, erscheinen. Im Programm muß dann ein Filter enthalten sein, der eine Fehlermeldung bei Falscheingaben (zum Beispiel Zeile = 234 oder ähnliches) ausgibt. Wir sind eigentlich schon mitten in der Besprechnung des ersten Teils unseres Maschinenprogrammes, nämlich der Parameterübergabe. Der zweite Teil muß nun aus den Angaben Zeile und Spalte den Ort in der Bit-Map ausrechnen, an den der String geschrieben wird. Das Startbyte in der Bit-Map ergibt sich (siehe Grafik-Folge 3) nach:

Startbyte = 320*Zeile + 8*Spalte + Anfangsadresse der Bit-Map

Nachdem das Startbyte bekannt ist, wird der String Zeichen für Zeichen durchgesehen, der ASCII-Code in den Bildschirmcode umgerechnet und schließlich in die Bit-Map eingeschrieben.

Das Umrechnen geschieht in einem kleinen Unterprogramm. Wieso eigentlich »Bildschirmcode«? Das liegt daran, daß der Bildschirmcode gleich der laufenden Nummer der Zeichen im Zeichen-ROM ist. Wie diese Zeichen dort aussehen, hatten wir uns schon in der 2. Folge der Grafik-Serie angesehen.

Auch das Einschreiben in die Bit-Map geschieht in einem Unterprogramm. Wo holen wir die Zeichen hier, wenn wir nicht bereit sind, das Zeichen-ROM ins RAM zu kopieren? Aus dem Zeichen-ROM selbst. Um das direkt lesen zu können, muß jeweils der Prozessorport ($01) so geschaltet werden, daß das Zeichen-ROM zugreifbar wird. Dr. H. Hauck hat's in seiner »Memory Map mit Wandervorschlägen«, 64'er, Ausgabe 11(1984) Seite 136 gut erklärt: Man erreicht das durch Löschen des Bit 2 im Prozessorport. Doch nun genug der Überlegungen, schreiben wir unser Programm!

Aus programmtechnischen Gründen taucht hier zuerst die Fehlerbehandlung auf:

```
8A54  LDX #0E     Fehlercode für ILLEGAL QUANTITY
8A56  JMP A437    Interpreter-Routine für die Ausgabe einer
                  Fehlermeldung, deren Code im X-Register
                  enthalten ist.
```

Hier fängt nun unser eigentliches Programm an mit der Übernahme der Parameter. Zunächst erfassen wir den String:

```
8A59  JSR AEFD    Interpreter-Routine, die auf Komma prüft.
8A5C  JSR AD9E    Interpreter-Routine, die einen Ausdruck auswertet. Wenn der Ausdruck ein String ist,
                  wird in $64/65 ein Zeiger auf den Stringdeskriptor eingerichtet.
```

64/65 ist eine häufig benutzte Speicheradresse. Wir lesen daher den Stringdeskriptor und lagern die Stringlänge in $24, die Stringstartadresse nach $04/05:

```
8A5F  LDY #00     Zähler auf Null
8A61  LDA (64),Y  Stringlänge in Akku
8A63  STA 24      und nach $24
8A65  INY         Zähler erhöhen
8A66  LDA (64),Y  LSB des Stringzeigers in Akku
8A68  STA 04      und nach $04
8A6A  INY         Zähler erhöhen
8A6B  LDA (64),Y  MSB des Stringzeigers in Akku
8A6D  STA 05      und nach $05
```

Damit ist der String gesichert, nun lesen wir die Zeilennummer:

```
8A6F  JSR AEFD    kennen wir schon: Auf Komma prüfen.
8A72  JSR AD9E    kennen wir ebenfalls, kann aber noch mehr
                  als nur Strings auszuwerten. Hier erkennt
                  diese Routine, daß eine Zahl vorliegt und
                  packt diese in den FAC (Fließkomma-
                  Akkumulator 1)
8A75  JSR B1AA    Interpreter-Routine: Wandelt den FAC-Inhalt
                  in eine 2-Byte-Integer-Zahl um. MSB landet
                  im Akku, LSB im Y-Register. Das MSB brauchen wir nicht.
8A78  CPY #19     Ist Zeile größer oder gleich dezimal 25?
8A7A  BCS 8A54    Wenn ja, Sprung zur Fehlermeldungsausgabe
```

Zugegeben, wenn das Programm hier gelandet ist, können sich immer noch einige Falscheingaben durchgeschmuggelt haben. Aber wer wird bei der Zeilennummer-Eingabe zum Beispiel eine negative Zahl wählen! Wenn Sie Lust haben, dann können Sie ja auch noch andere Fehlereingabe-Filter einbauen. Uns soll's so erst mal genügen. Weil wir die Zahl jetzt gerade in so schön greifbarer Form haben, berechnen wir auch gleich noch den Teil »320*Zeile« für die Position in der Bit-Map. Für diese Multiplikation verwenden wir eine Hires-3-Routine:

```
8A7C  STY 5B      Zeile nach $5B
8A7E  LDA #40     LSB der Zahl 320
8A80  STA 59      nach $59
8A82  LDA #01     MSB der Zahl 320
8A84  STA 5A      nach $5A
8A86  JSR 9410    Hires-3-Routine, die eine in $59/5A liegende
                  Zahl mit einer in $5B liegenden multipliziert.
                  Das Ergebnis findet man in $57/58.
```

320*Zeile ist nun in $57/58 gespeichert und wir übernehmen die Spaltenangabe ins Programm:

```
8A89  JSR AEFD    Wie gehabt: Komma prüfen
8A8C  JSR AD9E    Kennen wir auch schon
8A8F  JSR B1AA    Bekannt: Bringt Spalte ins Y-Register
8A92  CPY #28     Ist die Zahl größer oder gleich dezimal 40?
8A94  BCS 8A54    Wenn ja, dann Sprung zur Fehlermeldungsausgabe.
```

Hier gilt dasselbe, was für Zeile oben gesagt wurde. Und auch hier rechnen wir gleich den Ausdruck »8*Spalte« aus:

```
8A96  TYA         Spalte in Akku
8A97  CLC         Von hier an erfolgt die
8A98  ROL         Multiplikation mit 8
8A99  ROL         und das Ergebnis landet
8A9A  ROL         in den Speicherstellen
8A9B  STA 25      $25 (LSB)
8A9D  LDA #00     und
8A9F  ROL         $26 (MSB)
8AA0  STA 26
```

Nun addieren wir die beiden Ausdrücke (320*Zeile+8*Spalte):

```
8AA2  CLC
8AA3  LDA 57      LSB von 320*Zeile
8AA5  ADC 25      +LSB von 8*Spalte
8AA7  STA 25      nach $25
8AA9  LDA 58      MSB von 320*Zeile
8AAB  ADC 26      +MSB von 8*Spalte + Carry
8AAD  STA 26      nach $26
```

Schließlich zählen wir noch die Bit-Map-Startadresse dazu:

```
8AAF  CLC
8AB0  LDA 25      LSB von 320*Zeile+8*Spalte
8AB2  ADC #00     +LSB Bit-Map-Start
8AB4  STA 25      Ergebins=LSB Stringstart in der Bit-Map
                  nach $25
8AB6  LDA 26      MSB von 320*Zeile+8*Spalte
8AB8  ADC #A0     +MSB Bit-Map-Start
8ABA  STA 26      MSB des Stringstarts in der Bit-Map nach $26
```

Damit haben wir sowohl die Parameterübergabe als auch die Positionierung in der Bit-Map programmiert:
Wir finden nun in
$24 die Stringlänge,
$04/05 die Startadresse des Strings im Speicher
$25/26 die Startadresse des Strings in der Bit-Map
Wir kommen nun zu dem Programmteil, in dem der String Zeichen für Zeichen gelesen, umgerechnet und schließlich gedruckt wird:

8ABC	**LDY #00**	Zähler auf Null
8ABE	**LDA (04),Y**	String-Zeichen in den Akku lesen,
8AC0	**TAX**	und ins X-Register schieben
8AC1	**TYA**	Zähler in den Akku
8AC2	**PHA**	und auf den Stapel retten
8AC3	**TXA**	Akku-Inhalt wiederherstellen
8AC4	**JSR 8AD2**	Unterprogramm, das die Umrechnung des ASCII-Codes im Akku zum Bildschirmcode vornimmt
8AC7	**JSR 8AF6**	Unterprogramm, welches das Übertragen der Zeichen aus dem Zeichen-ROM in die Bit-Map durchführt
8ACA	**PLA**	Zähler vom Stapel holen
8ACB	**TAY**	und wieder ins Y-Register schreiben
8ACC	**INY**	Zähler erhöhen
8ACD	**CPY 24**	Vergleich des Zählers mit der Stringlänge
8ACF	**BMI 8ABE**	Stringlänge erreicht? Wenn ja, dann...
8AD1	**RTS**	Programmende und zurück ins aufrufende Programm.

Nun kommen wir noch zu den beiden Unterprogrammen. Zunächst die Umrechnung vom ADCII- in den Bildschirmcode. Der Code des eingelesenen Zeichens befindet sich im Akku:

8AD2	**BPL 8AD7**	Liegt ein geSHIFTetes Zeichen vor? Dann ist nämlich Bit 7 gesetzt. Wenn Bit 7 nicht gesetzt ist, erfolgt der Sprung
8AD4	**JMP 8AE6**	ansonsten wird hier zur Routine für SHIFT-Zeichen gesprungen

Jetzt haben wir's also mit nicht geSHIFTeten Zeichen zu tun:

8AD7	**CMP #20**	haben wir es etwa mit Steuerzeichen zu tun?
8AD9	**BCC 8AF3**	Wenn ja, dann verzweigen wir
8ADB	**CMP #60**	liegen Grafikzeichen vor?
8ADD	**BCC 8AE3**	wenn nein, dann Sprung
8ADF	**AND #DF**	mit der Makse 1101 1111 wird Bit 5 gelöscht
8AE1	**BNE 8AE5**	unbedingter Sprung
8AE3	**AND #3F**	mit der Maske 0011 1111 werden die Bits 6 und 7 gelöscht
8AE5	**RTS**	Fertig mit den ungeSHIFTeten Zeichen. Rücksprung ins aufrufende Programm.

Im folgenden bearbeiten wir die SHIFT-Zeichen:

8AE6	**AND #7F**	Mit der Maske 0111 1111 wird Bit 7 gelöscht
8AE8	**CMP #7F**	liegt das Pi-Zeichen vor?
8AEA	**BNE 8AEE**	Wenn nicht, Sprung
8AEC	**LDA #5E**	wenn ja, dann Code für das Pi-Zeichen in den Akku
8AEE	**CMP #20**	liegt ein Steuerzeichen vor?
8AF0	**BCC 8AF3**	Wenn ja, Sprung
8AF2	**RTS**	wenn nein, dann ist jetzt der Bildschirmcode im Akku und wir springen zurück zum aufrufenden Programm.

Nun haben wir es nur noch mit den Steuerzeichen zu tun. Die ignorieren wir und setzen dafür einfach ein Leerzeichen ein:

8AF3	**LDA #20**	Code für »Space« in Akku
8AF5	**RTS**	und Rücksprung zum aufrufenden Programm.

Kommen wir nun zum zweiten Unterprogramm, das die Zeichen, welche im Akku enthalten sind als Bildschirmcodes, aus dem Zeichen-ROM heraus und in die richtige Stelle der Bit-Map hineinliest:

8AF6	**LDX #00**	
8AF8	**STX 27**	LSB der Zeichen-ROM-Startadresse = 0
8AFA	**STX 29**	Zwischenspeicher auf Null
8AFC	**LDX #D0**	MSB-Zeichen-ROM-Startadresse
8AFE	**STX 28**	nach $28
8B00	**CLC**	Von hier an wird der
8B01	**ROL**	Zeichencode im Akku
8B02	**ROL 29**	mal 8 gerechnet
8B04	**ROL**	Zu guter Letzt findet
8B05	**ROL 29**	man das LSB des
8B07	**ROL**	Ergebnisses im Akku
8B08	**ROL 29**	und das MSB in $29
8B0A	**CLC**	Von hier an wird die
8B0B	**ADC 27**	Startadresse des
8B0D	**STA 27**	Zeichenmusters im ROM
8B0F	**LDA 28**	berechnet und in
8B11	**ADC 29**	$27/28 abgelegt
8B13	**STA 28**	

Damit wissen wir nun, daß 8 Bytes von der Adresse $27/28 im Zeichen-ROM zur Adresse $25/26 in der Bit-Map übertragen werden müssen. Das geschieht nun:

8B15	**LDY #00**	Y-Index auf Null
8B17	**LDX #08**	X-Register-Zähler auf 8
8B19	**LDA 01**	Prozessorport-Inhalt in Akku
8B1B	**PHA**	und auf den Stapel beiseitelegen
8B1C	**AND #FB**	mit Maske binär 1111 1011 Bit 2 löschen = Zeichen-ROM zugreifbar machen
8B1E	**SEI**	wir können jetzt keine Interrupts gebrauchen
8B1F	**STA 01**	den neuen Prozessorport-Inhalt einlesen
8B21	**LDA (27),Y**	das Zeichen-Muster Byte für Byte aus dem Zeichen-ROM herauslesen in Akku
8B23	**STA (25),Y**	und in Bit-Map einschreiben
8B25	**INY**	Y-Index erhöhen
8B26	**DEX**	X-Zähler vermindern
8B27	**BNE 8B21**	wiederholen, bis X-Zähler gleich Null
8B29	**PLA**	alten Prozessorport Inhalt vom Stapel zurückholen
8B2A	**STA 01**	und rekonstruieren
8B2C	**CLI**	jetzt darf wieder unterbrochen werden
8B2D	**CLC**	Ab hier wird die
8B2E	**LDA 25**	Zieladresse in der
8B30	**ADC #08**	Bit-Map um 8 erhöht
8B32	**STA 25**	$25/26 enthält dann
8B34	**LDA 26**	schon für das nächste
8B36	**ADC #00**	einzuschreibende
8B38	**STA 26**	Zeichen die aktuelle Adresse.
8B3A	**RTS**	Ende des Unterprogrammes. Rücksprung ins aufrufende Programm.

Damit hätten wir's. Als Programm 2 finden Sie — falls Sie ohne Assembler (zum Beispiel SMON) arbeiten — ein mittels MSE eintippbares Listing dieser Routine und für den Überblick ist als Bild 5 noch ein komplettes Flußdiagramm gezeigt.

Wir setzen das Puzzle zusammen

Um nun diese beiden Programmteile in Hires-3 einzubinden, sollten zunächst Programm 1 und Programm 2 abgetippt und gespeichert werden. Anschließend laden Sie Hires-3 (mit Load"HIRES-3", 8,1 beziehungsweise ,1,1 bei Kassettenbetrieb), geben die Schutz-POKEs ein:
POKE52,128:POKE56,128 und anschließend NEW. Das wurde in der Folge 8 der Grafikserie versehentlich ausgelassen. Nun laden Sie ebenfalls absolut (also mit LOAD"PROGRAMM 1",8,1 oder ,1,1) das Programm 1 ein, geben wieder NEW ein, laden dann absolut (!) das Programm 2 ein und schließen das alles mit einem letzten NEW ab. Hires-3 und die beiden Ergänzungen stehen nun nahtlos aneinandergefügt im Speicher. Um den TEX-Befehl zu ermöglichen, muß nun noch mittels einiger POKEs Hires-3 etwas verändert werden. Geben Sie also bitte noch ein:
POKE37694,89:POKE37695,138
POKE37858,84:POKE37859,69:POKE37860,88:POKE37861,0:POKE37862,0

Mit Hilfe des SMON oder eines anderen dazu fähigen Monitors können Sie das so ergänzte Programm Hires-3 nun komplett abspeichern, zum Beispiel beim SMON mit dem Kommando: S"HIRES-3", 08,8000,9DCB

Es wird Zeit, Hires-4 zu entwickeln. In der Befehlsliste von Hires-3 ist nämlich kein Byte mehr Platz gebleiben, um alle Optionen, die nun mit SYS-Kommandos aufgerufen werden, durch neue Befehlsworte anzusprechen. TEX war das letzte neue Wort, das gerade noch hineinpaßte. So rufen Sie nun alles Neue auf:

SYS 35256 Bildschirmaufspaltung durch Rasterzeileninterrupt anschalten.

SYS 35377 Bildschirmaufspaltung wieder ausschalten. Diesen SYS-Befehl müssen Sie sich gut merken. Wenn mitten im Programm der Computer bei aufgespaltenem Bild durch einen Fehler aussteigt, können Sie nämlich durch dieses SYS 35377 und anschließendes HOF wieder in den normalen Modus gelangen.

TEX,String,Zeile,Spalte Einschreiben eines durch String definierten Textes an die Stelle Zeile, Spalte in die Bit-Map. So setzt der Befehl TEX,"HALLO",10,12 den Text HALLO in die 10. Zeile ab Spalte 12.

Die Bildschirmaufspaltung sollte nicht zusammen mit dem UHR-Befehl und der Hardcopy-Routine betrieben werden. Hires-3 ist nämlich noch nicht darauf eingerichtet, mehrere Interrupt-Routinen parallel zu verarbeiten. Als Programm 3 finden Sie noch ein Basic-Demonstrationsprogramm, das einige Anwendungen der neuen Befehle erläutert. (Heimo Ponnath/gk)

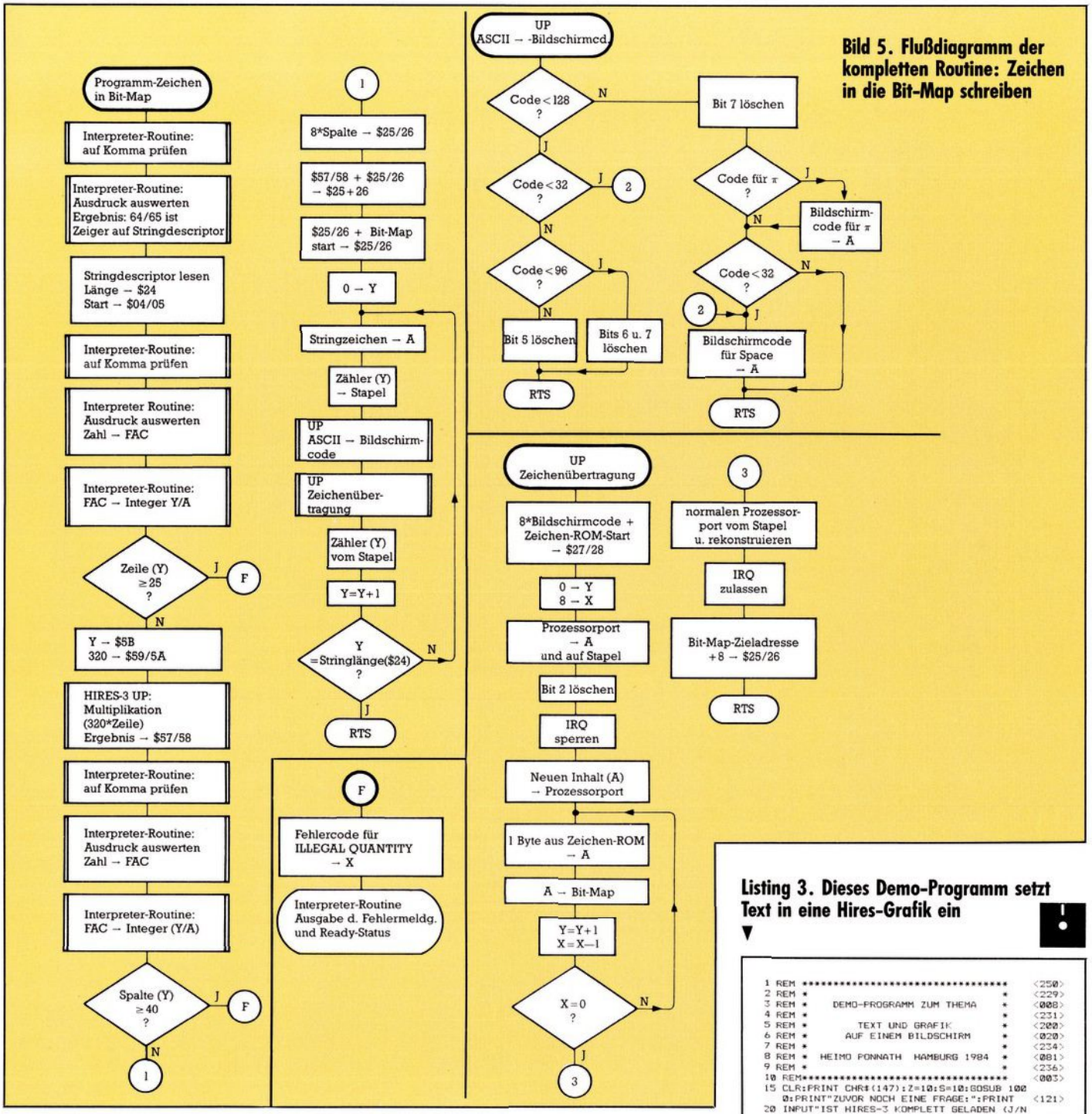

Bild 5. Flußdiagramm der kompletten Routine: Zeichen in die Bit-Map schreiben

Listing 1. Die Interrupt-Routine. Beachten Sie die Eingabe-Hinweise auf Seite 53/54 ▼

```
programm : programm1          89b8 8a53
-------------------------------------------
89b8 : a9 20 a2 27 9d 00 8c 9d   46
89c0 : 48 8f ca 10 f7 a2 77 9d   32
89c8 : 70 8f ca 10 fa 78 a9 ec   a8
89d0 : 8d 14 03 a9 89 8d 15 03   bd
89d8 : a9 3a 8d 12 d0 ad 11 d0   a4
89e0 : 29 7f 8d 11 d0 a9 81 8d   ca
89e8 : 1a d0 58 60 ad 19 d0 8d   8e
89f0 : 19 d0 30 07 ad 0d dc 58   c6
89f8 : 4c 31 ea ad 12 d0 c9 da   d2
8a00 : b0 1d a9 38 a0 3b 8d 18   fa
8a08 : d0 8c 11 d0 a2 27 ad 26   e3
8a10 : 8e 9d 20 8f ca 10 fa a9   d3
8a18 : da 8d 12 d0 4c 81 ea a9   27
8a20 : 34 a0 1b 8d 18 d0 8c 11   79
8a28 : d0 a9 3a 8d 12 d0 4c 81   e9
8a30 : ea 78 a9 00 8d 1a d0 a9   01
8a38 : 31 8d 14 03 a9 ea 8d 15   e8
8a40 : 03 58 a9 38 a0 3b 8d 18   2b
8a48 : d0 8c 11 d0 ad 26 8e 20   43
8a50 : 32 95 60 ff 00 ff 00 ff   65
```

Listing 2. Direktes Einschreiben von Text in die Bit-Map ►

```
programm : programm2          8a54 8b3b
-------------------------------------------
8a54 : a2 0e 4c 37 a4 20 fd ae   98
8a5c : 20 9e ad a0 00 b1 64 85   75
8a64 : 24 c8 b1 64 85 04 c8 b1   e4
8a6c : 64 85 05 20 fd ae 20 9e   eb
8a74 : ad 20 aa b1 c0 19 b0 d8   5b
8a7c : 84 5b a9 40 85 59 a9 01   ec
8a84 : 85 5a 20 10 94 20 fd ae   e0
8a8c : 20 9e ad 20 aa b1 c0 28   f6
8a94 : b0 be 98 18 2a 2a 2a 85   74
8a9c : 25 a9 00 2a 85 26 18 a5   10
8aa4 : 57 65 25 85 25 a5 58 65   54
8aac : 26 85 26 18 a5 25 69 00   4b
8ab4 : 85 25 a5 26 69 a0 85 26   f8
8abc : a0 00 b1 04 aa 98 48 8a   ef
8ac4 : 20 d2 8a 20 f6 8a 68 a8   ab
8acc : c8 c4 24 30 ed 60 10 03   2e
8ad4 : 4c e6 8a c9 20 90 18 c9   ea
8adc : 60 90 04 29 df d0 02 29   89
8ae4 : 3f 60 29 7f c9 7f d0 02   6d
8aec : a9 5e c9 20 90 01 60 a9   21
8af4 : 20 60 a2 00 86 27 86 29   fb
8afc : a2 d0 86 28 18 2a 26 29   6b
8b04 : 2a 26 29 2a 26 29 18 65   a8
8b0c : 27 85 27 a5 28 65 29 85   d2
8b14 : 28 a0 00 a2 08 a5 01 48   23
8b1c : 29 fb 78 85 01 b1 27 91   6f
8b24 : 25 c8 ca d0 f8 68 85 01   65
8b2c : 58 18 a5 25 69 08 85 25   d6
8b34 : a5 26 69 00 85 26 60 ff   52
```

Listing 3. Dieses Demo-Programm setzt Text in eine Hires-Grafik ein ▼

```
1 REM ********************************    <250>
2 REM *                              *    <229>
3 REM *    DEMO-PROGRAMM ZUM THEMA   *    <008>
4 REM *                              *    <231>
5 REM *       TEXT UND GRAFIK        *    <200>
6 REM *     AUF EINEM BILDSCHIRM     *    <020>
7 REM *                              *    <234>
8 REM *  HEIMO PONNATH  HAMBURG 1984 *    <081>
9 REM *                              *    <236>
10 REM********************************    <003>
15 CLR:PRINT CHR$(147):Z=10:S=10:GOSUB 100
   0:PRINT"ZUVOR NOCH EINE FRAGE:":PRINT    <121>
20 INPUT"IST HIRES-3 KOMPLETT GELADEN (J/N
   )";A$                                    <192>
25 IF A$="N"THEN PRINT:PRINT CHR$(18)"BRAU
   CHEN SIE ABER!"CHR$(146):END             <246>
30 POKE 52,128:POKE 56,128:SYS 37498:PRINT
    CHR$(147)                               <084>
40 REM ++++++ SINUSKURVE ZEICHNEN +++++     <003>
45 DEF FN A(X)=SIN(X):XU=-2*π:XO=2*π:YU=-2
   .5:YO=2:TRS,XU,XO,YU,YO:HFL,6,14         <049>
50 FUNKT,A,XU,XO:TLN,XU,0,XO,0:TLN,0,YU,0,
   YO:REC,0,0,319,199                       <081>
55 REM ++++++ DER TEX-BEFEHL ++++++++++     <021>
60 TEX,"DIES IST EINE SINUSKURVE",3,8       <181>
65 REM +++++ BILDSCHIRMAUFSPALTUNG ++++     <126>
70 SYS 35256:SYS 34647:Z=21:S=0:GOSUB 1000
   :PRINT"WUENSCHEN SIE EINE SKALIERUNG";    <098>
75 INPUT"(J/N)";A$:IF A$="N"THEN 115        <120>
80 REM +++++++ SKALIERUNG  ++++++++++++     <005>
85 DEF FN X(X)=INT(39*(X+2*π)/(4*π)):DEF F
   N Y(Y)=INT(24*(2-Y)/4.5)                 <123>
90 FOR X=-6 TO 6:TLN,X,0,X,-.1:X$=STR$(X):
   A=FN Y(-.3):B=FN X(X)                    <147>
95 TEX,X$,A,B:NEXT X                        <231>
100 TLN,0,1,-.2,1:TLN,0,-1,-.2,-1:A=FN Y(1
    ):B=FN X(.5):TEX,"1",A,B:A=FN Y(-1)     <245>
105 B=FN X(.3):TEX,"-1",A,B:CLR:INPUT"HARD
    COPY (J/N)";A$:IF A$="J"THEN GOSUB 200
    0                                       <043>
110 REM +++++ PROGRAMMENDE ++++++++++++     <089>
115 Z=21:S=0:GOSUB 1000:PRINT"GEBEN SIE NA
    CH READY EIN:SYS35577:HOF "             <246>
120 END                                     <248>
999 REM ++++++ UP CURSOR SETZEN +++++++     <004>
1000 POKE 214,Z:POKE 211,S:SYS 58640:RETUR
     N                                      <042>
1999 REM +++++++ UP HARDCOPY ++++++++++     <063>
2000 SYS 35577:OPEN 1,4,10:PRINT#1:CLOSE 1
     :SYS 34865:SYS 35256:RETURN            <010>
```