Tips und Tricks für Basic-Programmierer

Mit den folgenden kleinen Programmen geben wir Ihnen einige Hilfsmittel an die Hand, die Sie in Ihre Programme einbinden können, um sie schneller und effektiver zu machen.

Vielleicht geht es Ihnen wie mir. Sie kommen morgens nicht aus Ihrem Bett, verfluchen jeden Tag aufs neue Ihren Wecker, oder verbrauchen die Nerven Ihres Partners, der verzweifelt versucht, Sie zu wecken. Falls dies der Fall sein sollte, haben Sie sicher schon einmal versucht, sich durch Ihren Computer wecken zu lassen. Sollten Sie zu diesem Zweck die Variable TI$ (die sich ja geradezu aufdrängt) verwendet haben, so würden Sie zwar an diesem Morgen geweckt, jedoch sicher nicht zu dem Zeitpunkt, den Sie sich vorgestellt hatten. Die Variable TI$ hat leider die unangenehme Eigenschaft, eine recht ungenaue Uhr zu sein. Der maximale Fehler liegt bei 30 Minuten pro Tag. Aber zum Glück hat unser Computer, wie so oft, auch hierfür eine Lösung parat. Es handelt sich hierbei um die beiden CIAs (Complex Interface Adapter), welche je eine Echtzeituhr enthalten. Diese Echtzeituhr erhält ihren Takt aus der Netzfrequenz, die Variable TI$ hingegen wird von der Interruptroutine versorgt. Dieser Tatsache verdanken wir es, daß die CIA-Uhr eine so große Langzeitgenauigkeit vorweisen kann. Die Echtzeituhr hält aber noch eine weitere Überraschung für uns bereit. Es besteht die Möglichkeit, eine Alarmzeit anzugeben. Sobald diese Zeit erreicht wird, löst das CIA einen IRQ (Interrupt, beziehungsweise CIA 2 einen NMI = unmaskierter Interrupt) aus, aber dazu später noch mehr. Wie die Register der CIA-Bausteine belegt sind, können Sie aus Tabelle 1 entnehmen. Die Echtzeituhr belegt die Register 8 bis 11.

Die Zeit starten

Die CIA-Uhren sind im BCD-Format organisiert. Dadurch sparen wir uns zwar in Maschinensprache das lästige Umrechnen, von Basic aus ist die Umwandlung leider nicht zu umgehen. Im BCD-Format werden Zahlen zwar immer noch Bitweise gespeichert, jedoch jede Ziffer einzeln. Mit 8 Bit lassen sich also zwei Ziffern (2 mal 4 Bit, 00 bis 99) darstellen. Um nun eine Zahl, bestehend aus zwei Ziffern, in das BCD-Format umzuwandeln, wird die erste Ziffer (Wertigkeit 10) mit 16 multipliziert und zu der zweiten Ziffer (Wertigkeit 1) addiert.
Aber nun wieder zu unseren CIAs. Die Basisadresse des CIA 1 ist 56320 (CIA 2 = 56576). Eine kurze Registerbeschreibung entnehmen Sie bitte Bild 1. Zum Stellen der Uhr werden einfach die entsprechenden BCD-Werte in die jeweiligen Register geschrieben. Sobald Sie das $^1/_{10}$ Sekunden-Register überschreiben, startet die Uhr (überschreiben Sie jedoch die Stundenregister, stoppt sie). An Bit 7 in Register 11 können Sie außerdem erkennen, ob es sich um eine AM- oder PM-Zeit handelt (AM = Vormittag, PM = Nachmittag). Bitte vergessen Sie nicht, Bit 7 in Register 14 zu setzen (Netzfrequenz 50 HZ (60 HZ)). Für den Fall, daß Sie Ihre Uhr auch lesen möchten (soll ja vorkommen), bietet das CIA eine weitere Besonderheit. Sobald Sie eines der Register lesen, wird die komplette Uhrzeit in einen Zwischenspeicher übernommen. Durch diesen Kniff können Sie in aller Ruhe die Daten auslesen, ohne daß eine Veränderung des Zwischenspeichers eintritt. Sobald Register 8 angesprochen wird, gibt das CIA den Speicher wieder frei. In Listing 1 wartet ein Basic-Programm auf Sie, mit welchem Sie die Zeit stellen und ablesen können (Start mit RUN, um sie zu stellen und RUN 200, um sie zu stoppen). Listing 2 enthält eine kurze Assemblerroutine, die in die Interruptroutine eingefügt wird und Ihnen die Uhrzeit laufend auf dem Bildschirm ausgibt. Die eigentliche Assemblerroutine liegt bei Adresse 830. Um das Programm zu aktivieren, befindet sich bei Adresse 900 ein Programmteil, welcher den Interruptvektor auf das Hauptprogramm legt (SYS 900 startet die Anzeige der Uhr).

Der Wecker im Computer

Neben einer Uhr hat das CIA natürlich noch weitere Aufgaben. Es löst zum Beispiel das Interruptsignal aus, durch welches die bereits erwähnte Interruptroutine angesprungen wird. Dieses Ereignis tritt bei Unterlauf des Timers (die CIAs haben neben der Uhr noch zwei Timer) automatisch alle $^1/_{60}$ Sekunden auf. Wir haben allerdings die Möglichkeit zu einem von uns vordefinierten Zeitpunkt einen zusätzlichen Interrupt auszulösen. Zu diesem Zweck müssen wir eine Zeit angeben, die wir als Alarmzeit heranziehen. Die Alarmzeit wird wie die normale Uhrzeit in die Register 8 bis 11 geschrieben, jedoch muß hierfür das Bit 7 in Register 15 gesetzt sein (Rücksetzen nicht vergessen). Stimmen nun die Uhrzeit und die von uns eingestellte Alarmzeit überein, wird ein zusätzlicher IRQ ausgelöst und der Programmzeiger springt zu der

CIA (6526)

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
Register 0	Port A								
$DC00	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	56320
Register 1	Port B								
$DC01	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	56321
Register 2	Datenrichtungsregister A								
$DC02	DPA7	DPA6	DPA5	DPA4	DPA3	DPA2	DPA1	DPA0	56322
Register 3	Datenrichtungsregister B								
$DC03	DPB7	DPB6	DPB5	DPB4	DPB3	DPB2	DPB1	DPB0	56323
Register 4	Timer A Low-Byte								
$DC04	TAL7	TAL6	TAL5	TAL4	TAL3	TAL2	TAL1	TAL0	56324
Register 5	Timer A High-Byte								
$DC05	TAM7	TAM6	TAM5	TAM4	TAM3	TAM2	TAM1	TAM0	56325
Register 6	Timer B Low-Byte								
$DC06	TBL7	TBL6	TBL5	TBL4	TBL3	TBL2	TBL1	TBL0	56326
Register 7	Timer B High-Byte								
$DC07	TBM7	TBM6	TBM5	TBM4	TBM3	TBM2	TBM1	TBM0	56327
Register 8	Uhr 1/10 Sekunde								
$DC08	0	0	0	0	T0	T4	T2	T1	56328
Register 9	Uhr Sekunde								
$DC09	0	SM4	SM2	SM1	SL0	SL4	SL2	SL1	56329
Register 10	Uhr Minuten								
$DC0A	0	MH4	MH2	MH1	ML0	ML4	ML2	ML1	56330
Register 11	Uhr Stunden								
$DC0B	PM	0	0	MH	ML0	ML4	ML2	ML1	56331
Register 12	Serial Data Register								
$DC0C	S7	S6	S5	S4	S3	S2	S1	S0	56332
Register 13	Interrupt Control Register								
$DC0D	IRQ	0	0	FLG	SP	PLRM	T0	TA	56333
Register 14	Control Register A								
$DC0E	S0 HZ	SP MODE	IN MODE	LOAD	RUN MODE	OUT MODE	PB ON	START	56334
Register 15	Control Register B								
$DC0F	ALARM	IN MODE	IN MODE	LOAD	RUN MODE	OUT MODE	PB ON	START	56335

Tabelle 1. Registerbelegung der CIA

in den IRQ-Vektoren angegebenen Adresse. Die Unterscheidung zwischen einem Interrupt durch den Unterlauf eines Timers oder durch Erreichen der Alarmzeit wird durch Register 13 möglich. In diesem Register wird bei Auftreten des IRQ das zuständige Bit (siehe Tabelle 1) gesetzt. Zum besseren Verständnis erwartet Sie in Listing 3 eine kleine Routine, welche bei Erreichen der Alarmzeit die Farbe des Bildschirmrahmens verändert. Diese Routine wird ebenfalls mit SYS 900 gestartet.

Bildschirm mal zwei

Wäre es nicht praktisch, auf dem Monitor zwei voneinander unabhängige Bildschirmseiten unterzubringen? Würde sich ein HiRes-Bild und der dazugehörige Text nicht ungeheuer gut machen? Wenn Ihnen diese oder ähnliche Gedanken schon einmal durch den Kopf gingen, dann sind Sie hier genau richtig. Mittels des VICs (Videocontroller) ist es unserem Computer nicht nur möglich, Zahlen oder Buchstaben auf den Bildschirm zu bringen, er sorgt auch für die HiRes-Grafik, die Sprites und die Verwaltung von 16 KByte Speicherplatz. Da er nebenbei auch noch für die Erzeugung eines normgerechten PAL-Signals zuständig ist, muß er in jedem Augenblick genau wissen, an welcher Stelle sich der Elektronenstrahl Ihres Monitors gerade befindet. Mit einer kleinen Assemblerroutine und ein wenig Bastelei an den Interruptvektoren und -registern unseres Computers können Sie in dem Augenblick, in dem sich der Elektronenstrahl an einer von Ihnen bestimmten Position befindet, zum Beispiel von einem HiRes- auf einen Textbildschirm umschalten (und umgekehrt). Da wir dadurch dem Computer keine Chance lassen, eine Bildschirmseite zu vollenden und einen Bildwechsel vorzunehmen, erscheinen zwei (oder mehrere) Bilder gleichzeitig. Die Trennung dieser Bilder ist abhängig von der Position, an welcher die Umschaltung erfolgt. Die Assemblerroutine in Listing 4 ermöglicht die Darstellung von zwei Textbildschirmen (1024-2023 und 2024-3043) gleichzeitig. In der Speicherstelle 648 können Sie festlegen, auf welchem Bildschirmteil Ihre Eingaben erscheinen sollen. Alle Ausgaben auf den Bildschirm werden ebenfalls in die Bildschirmseite geschrieben, deren Adresse Sie in Speicherstelle 648 angegeben haben. Der Wert, den Sie an diese Adresse POKEn müssen, errechnet sich wie folgt: Adresse des Bildschirmspeichers durch 256. Sobald Sie sich für eine Bildschirmhälfte entschieden haben, können Sie das Programm mit SYS 900 starten (Hier wird der Interruptvektor gesetzt, das eigentliche Programm befindet sich ab Speicherplatz 830. Oberer Bildschirm POKE 648,4, unterer Bildschirm POKE 648,8).

Der VIC hält für uns die Möglichkeit bereit, bei Auftreten eines bestimmten Ereignisses einen Interrupt (IRQ) auszulösen. Zu diesem Zweck stehen die Register 25 und 26 zur Verfügung. Die genaue Belegung dieser Register entnehmen Sie bitte Tabelle 2. Falls mindestens ein Bit aus Register 25 und Register 26 übereinstimmt, wird ein IRQ ausgelöst und der interne Programmzähler springt zu der im IRQ-Vektor (788/789) angegebenen Adresse. Sobald wir diesen Zeiger auf ein eigenes Programm zeigen lassen, wird bei jedem IRQ zuerst unser Programm angesprungen. Das Ende unseres Programmes sollte immer einen unbedingten Sprung in die eigentliche Interruptroutine beinhalten. In unserem Fall müssen wir die Interruptmaske (Register 26) so verändern, daß Register 18 als Auslöser anerkannt wird. Um eine Auslösung an einer bestimmten Stelle des Rasterstrahls zu erreichen, wird in Register 18 (17) die entsprechende Position angegeben. Register 18 hat folglich zwei Aufgaben: 1. wird es gelesen, so gibt es die aktuelle Position des Rasterstrahls an. 2. wird es beschrieben, so gilt der Wert, welcher in das Register geschrieben wurde als die Position des Strahls, an der der IRQ ausgelöst (und Bit 1/Register 25 ist gesetzt) wird. Als letztes Problem bleibt uns noch das CIA. Dieser Chip ist im Normalfall derjenige, welcher die IRQs auslöst. Für unseren Zweck benötigen wir jedoch nur das Interruptsignal, das vom VIC ausgelöst wurde. Aus diesem Grund wird zu Beginn der

```
10 C=56328                                          <226>
20 POKE C+14,PEEK(C+14) OR 128                      <163>
30 POKE C+15,PEEK(C+15) AND 127                     <136>
40 INPUT "ZEIT HHMMSS";A$                           <220>
50 IF LEN(A$)<>6 THEN 40                            <057>
60 H=VAL(LEFT$(A$,2))                               <026>
70 M=VAL(MID$(A$,3,2))                              <239>
80 S=VAL(RIGHT$(A$,2))                              <154>
100 IF H>12 THEN H=H+68                             <001>
110 POKE C+3,16*INT(H/10)+H-INT(H/10)*10            <211>
130 POKE C+2,16*INT(M/10)+M-INT(M/10)*10            <108>
150 POKE C+1,16*INT(S/10)+S-INT(S/10)*10            <090>
160 POKE C,0                                        <224>
180 PRINT"{CLR}"                                    <168>
200 C=56328                                         <160>
210 H=PEEK(C+3):M=PEEK(C+2):S=PEEK(C+1):T=
    PEEK(C)                                         <230>
230 F=0:IF H>32 THEN H=H-128:F=1                    <013>
240 H=INT(H/16)*10+H-INT(H/16)*16:IF F=0 T
    HEN 280                                         <251>
250 IF H=12 THEN 290                                <126>
260 H=H+12                                          <250>
280 IF H=12 THEN H=0                                <167>
290 M=INT(M/16)*10+M-INT(M/16)*16                   <173>
300 S=INT(S/16)*10+S-INT(S/16)*16                   <131>
320 T$=RIGHT$("0"+MID$(STR$(H),2,2),2)+":"          <254>
340 T$=T$+RIGHT$("0"+MID$(STR$(M),2,2),2)+
    ":"                                             <023>
360 T$=T$+RIGHT$("0"+MID$(STR$(S),2,2),2)+
    ":"                                             <067>
380 T$=T$+RIGHT$(STR$(T),1)                         <166>
400 PRINT"{HOME,3DOWN}";T$                          <050>
420 GOTO 210                                        <118>
```

Listing 1. Basic-Programm zum Stellen der Uhr

```
NAME : LISTING 2 MC     033E 03A3
-------------------------------------
033E : A2 03 A0 1E BD 08 DC 85  E8
0346 : FB 29 F0 18 6A 6A 6A 6A  8D
034E : 69 30 EA 99 00 04 A5 FB  6C
0356 : 29 0F 69 30 C8 99 00 04  C9
035E : C8 A9 3A 99 00 04 C8 CA  95
0366 : E0 00 D0 D8 AD 08 DC 69  F7
036E : 2F 99 00 04 4C 31 EA EA  BA
0376 : EA EA EA EA EA EA EA EA  75
037E : EA EA EA EA EA EA 78 A9  30
0386 : 3E 8D 14 03 A9 03 8D 15  03
038E : 03 58 60 EA EA EA EA EA  BA
0396 : EA EA EA E6 FC D0 F4 E6  85
039E : FD A5 FD C9 08 00 00 00  A7
```

Listing 2. Interruptroutine Uhrzeit

```
NAME : LISTING 3 MC     033E 03A3
-------------------------------------
033E : A2 03 A0 1E BD 08 DC 85  E8
0346 : FB 29 F0 18 6A 6A 6A 6A  8D
034E : 69 30 EA 99 00 04 A5 FB  6C
0356 : 29 0F 69 30 C8 99 00 04  C9
035E : C8 A9 3A 99 00 04 C8 CA  95
0366 : E0 00 D0 D8 AD 08 DC 69  F7
036E : 2F 99 00 04 AD 0D DC C9  35
0376 : 81 F0 03 EE 20 D0 4C 31  2A
037E : EA EA EA EA EA EA 78 A9  30
0386 : 3E 8D 14 03 A9 03 8D 15  03
038E : 03 58 60 EA EA EA EA EA  BA
0396 : EA EA EA E6 FC D0 F4 E6  85
039E : FD A5 FD C9 08 00 00 00  A7
```

Listing 3. Interruptroutine Alarmzeit

```
NAME : LISTING 4 MC     033E 03AD
-------------------------------------
033E : AD 19 D0 A2 FF 8E 19 D0  7B
0346 : 29 80 C9 80 F0 06 4C 31  05
034E : EA EA EA EA AD 18 D0 C9  38
0356 : 15 F0 07 A9 15 A2 96 18  CB
035E : 90 04 A9 25 A2 00 8D 18  90
0366 : D0 8E 12 D0 4C B1 EA EA  6E
036E : EA EA EA EA EA EA EA EA  6D
0376 : EA EA EA EA EA EA EA EA  75
037E : EA EA EA EA EA EA 78 A9  30
0386 : 3E A2 03 8D 14 03 8E 15  46
038E : 03 A9 F9 8D 1A D0 AD 11  97
0396 : D0 29 7F 8D 11 D0 A9 96  F8
039E : 8D 12 D0 58 60 EA EA EA  52
03A6 : EA EA EA EA EA EA EA 00  CF
```

Listing 4. Bildschirmteilung

Register	Adresse dezimal	Adresse hex	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	53248	$D000	X-Position Sprite Nr. 0							
1	53249	$D001	Y-Position Sprite Nr. 0							
2	53250	$D002	X-Position Sprite Nr. 1							
3	53251	$D003	Y-Position Sprite Nr. 1							
4	53252	$D004	X-Position Sprite Nr. 2							
5	53253	$D005	Y-Position Sprite Nr. 2							
6	53254	$D006	X-Position Sprite Nr. 3							
7	53255	$D007	Y-Position Sprite Nr. 3							
8	53256	$D008	X-Position Sprite Nr. 4							
9	53257	$D009	Y-Position Sprite Nr. 4							
10	53258	$D00A	X-Position Sprite Nr. 5							
11	53259	$D00B	Y-Position Sprite Nr. 5							
12	53260	$D00C	X-Position Sprite Nr. 6							
13	53261	$D00D	Y-Position Sprite Nr. 6							
14	53262	$D00E	X-Position Sprite Nr. 7							
15	53263	$D00F	Y-Position Sprite Nr. 7							
16	53264	$D010	Sprite 7, msb X-Position	Sprite 6, msb X-Position	Sprite 5, msb X-Position	Sprite 4, msb X-Position	Sprite 3, msb X-Position	Sprite 2, msb X-Position	Sprite 1, msb X-Position	Sprite 0, msb X-Position
17	53265	$D011	msb des Rasterregisters (Reg. 18)	Schaltbit für veränderten Hintergrund-farbmodus 1 = eingeschaltet	Schaltbit für Hochauflösungs-modus 1 = eingeschaltet	Schaltbit für Schirm löschen 0 = gelöscht	Schaltbit für Zeilenzahl 0 = 24 Zeilen 1 = 25 Zeilen	Wert der Zeilenverschiebung in Y-Richtung beim Smooth Scrolling		
18	53266	$D012	Rasterregister. Dazu kommt das msb in Bit 7, Register 17							
19	53267	$D013	Lichtgriffel X-Position							
20	53268	$D014	Lichtgriffel Y-Position							
21	53269	$D015	Ein- und Ausschalten von Sprites. 0 = Sprite aus, 1 = Sprite an							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
22	53270	$D016	(unbenutzt)		Reset-Bit, muß 0 sein, damit VIC-II-Chip arbeitet	Schaltbild für Mehrfarbmodus 1 = eingeschaltet	Schaltbit für Spaltenzahl 0 = 38 Spalten 1 = 40 Spalten	Wert der Spaltenverschiebung in X-Richtung beim Smooth Scrolling		
23	53271	$D017	Sprite-Vergrößerung in Y-Richtung. 0 = normale Größe, 1 = doppelte Größe							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
24	53272	$D018	Startadresse Textbildschirm				Startadresse Zeichengenerator oder HiRes-Bitmap			(unbenutzt)
25	53273	$D019	Interrupt-Flag-Register				Lichtgriffel-Interrupt-Flag	Sprite/Sprite-Kollision	Sprite/Hinter-grund-Kollision	Raster-Interrupt-Flag
26	53274	$D01A	Interrupt-Masken-Register				Lichtgriffel-Interrupt	Sprite/Sprite-Kollision	Sprite/Hinter-grund-Kollision	Raster-Interrupt-Maske
27	53275	$D01B	Sprite/Hintergrund-Prioritätenregister. 0 = Sprite hat Priorität, 1 = Hintergrund hat Priorität							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
28	53276	$D01C	Sprite-Mehrfarbmodus-Register. 0 = Normaldarstellung, 1 = Mehrfarbmodus-Darstellung							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
29	53277	$D01D	Sprite-Vergrößerung in X-Richtung. 0 = normale Größe, 1 = doppelte Größe							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
30	53278	$D01E	Sprite/Sprite-Kollision. 0 = keine Berührung. 1 = Berührung							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
31	53279	$D01F	Sprite/Hintergrund-Kollision. 0 = keine Berührung. 1 = Berührung							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
32	53280	$D020	(unbenutzt)				Farbe des Bildschirmrahmens			
33	53281	$D021	(unbenutzt)				Hintergrundfarbe Nr. 0 (normale Hintergrundfarbe)			
34	53282	$D022	(unbenutzt)				Hintergrundfarbe Nr. 1			
35	53283	$D023	(unbenutzt)				Hintergrundfarbe Nr. 2			
36	53284	$D024	(unbenutzt)				Hintergrundfarbe Nr. 3			
37	53285	$D025	(unbenutzt)				Sprite-Mehrfarben-Register Nr. 0			
38	53286	$D026	(unbenutzt)				Sprite-Mehrfarben-Register Nr. 1			
39	53287	$D027	(unbenutzt)				Sprite 0, Farbe			
40	53288	$D028	(unbenutzt)				Sprite 1, Farbe			
41	53289	$D029	(unbenutzt)				Sprite 2, Farbe			
42	53290	$D02A	(unbenutzt)				Sprite 3, Farbe			
43	53291	$D02B	(unbenutzt)				Sprite 4, Farbe			
44	53292	$D02C	(unbenutzt)				Sprite 5, Farbe			
45	53293	$D02D	(unbenutzt)				Sprite 6, Farbe			
46	53294	$D02E	(unbenutzt)				Sprite 7, Farbe			

Tabelle 2. Alle Register des Video-Chips auf einen Blick

Routine mittels des IRQ-Flags in Register 25 abgefragt, ob der VIC den IRQ angefordert hat.

Basic-Zeilen in ein laufendes Programm einfügen? Wollen Sie Funktionen austesten oder ein Basic-Programm entwickeln, welches sich selbst verändert? Mit unserem Basic »Listing 5« und der Assemblerroutine aus »Listing 5 MC« können Sie ohne Schwierigkeiten alle beliebigen Daten in Ihr Basicprogramm aufnehmen, ohne das Programm stoppen zu

Nachträglich Befehle einfügen

müssen. Für diese Aufgabe stellt das Betriebssystem unseres Computers einige sehr effektive Routinen zur Verfügung. Unsere Aufgabe besteht nur noch darin, die einzufügenden Daten in den Basic-Eingabepuffer (512–600) zu schreiben und die Routine aufzurufen. Nach Aufruf des Programms wird der Vektor zur Eingabe einer Zeile auf eine Adresse innerhalb unseres Programms gesetzt. Daraufhin wird die komplette Betriebssystemroutine zur Einfügung von Programmzeilen abgearbeitet. Zum Ende dieser Routine würde der Programmzeiger in die Eingabe-Warteschleife springen. Durch die anfängliche Änderung des entsprechenden Vektors wird der Programmzeiger jedoch wieder auf unsere Routine umgelenkt. Hier wird nun der Vektor wieder auf den ursprünglichen Wert verändert und das Basic-Programm erneut gestartet. Wie bereits zu Anfang erwähnt, eignet sich dieser kleine Trick hervorragend zur Eingabe von Funktionen.

Basic optimieren

Sie haben sich sicher schon oft über die schwache Leistung unseres Basic-Interpreters geärgert. Je komplexer die Anforderungen an ein Basic-Programm sind, desto langweiliger wird die Geschwindigkeit der betreffenden Programme. Viele scheuen jedoch den Schritt in Richtung Maschinensprache und ärgern sich weiterhin mit ihren »Schlafmützen«-Programmen. Betrachtet man aber die Arbeitsweise des Interpreters ein wenig genauer, so bieten sich einige Möglichkeiten zur Beschleunigung von Basic-Programmen. Einmal abgesehen von den verwendeten Algorithmen und der Struktur des Programmes gibt es mehrere Tricks, um den Interpreter ein wenig anzutreiben.

Variable

Sobald in einem Basic-Programm eine Variable benötigt wird, beginnt der Interpreter, vom Variablenstart an aufwärts zählend, nach der Variable zu suchen. Je später eine Variable definiert wurde, desto länger dauert der Suchvorgang. Ähnlich verhält es sich bei Konstanten. Werden Konstante im Programm angegeben, so müssen diese bei jedem Durchlauf erneut umgewandelt werden. Definiert man Konstante als Variable, dann fällt die Umrechnung nur einmal an.

(1) Konstante, Variable, und Felder in der Reihenfolge ihrer Zugriffshäufigkeit vordefinieren. Möglichst ein- oder zweistellige Variablennamen verwenden.

Unterprogramme

Unterprogramme müssen vom Interpreter, wie Variablen, erst einmal gesucht werden. Da die Suche mit der niedrigsten Zeilennummer beginnt, findet er Zeilen am Anfang natürlich schneller.

(2) Unterprogramme sollten am Anfang eines Programmes stehen.

Verzweigungen

IF-THEN-Abfragen mit mehreren, durch AND-Verknüpfungen verbundenen, Bedingungen sollten durch Hintereinanderlegen von mehreren IF-THEN-Abfragen verschachtelt werden. Beispiel:

a. IF (A kleiner 100 AND B größer 0) THEN

b. IF A kleiner 100 THEN IF B größer 0 THEN

Version b bricht sofort nach Nichterfüllung der ersten Bedingung ab und ist somit schneller.

(3) IF-THEN-Abfragen mit mehreren Bedingungen verschachteln.

Leerzeichen

REMs und Leerstellen verzögern den Programmlauf, da der Interpreter sie ja ignorieren muß. Je mehr Befehle sich in einer Programmzeile befinden, desto seltener muß der Computer nach neuen Zeilenanfängen suchen.

(4) Programme kompakter verfassen.

Strings

Die Garbage-Collection-Routine des Betriebssystems beseitigt den »String-Müll«. Werden mehrere Strings definiert, fällt auch mehr »Müll« an. Das Tragische an dieser Routine ist die Geschwindigkeit: verdoppelt sich die Anzahl der Strings, vervierfacht (!) sich die Laufzeit. Aus diesem Grund bringt das Anlegen von String-Konstanten auch keinen Zeitgewinn.

(5) So wenig Strings wie möglich benutzen.

Potenzieren

Falls in Ihrem Programm eine ganzzahlige Potenzierung (x^2, x^4..) vorkommt, ist es ratsam, die Potenzierung durch eine Mehrfachmultiplikation zu ersetzen. Da die Potenzierungsroutine des Interpreters auch Potenzierungen mit Brüchen berechnen kann, ist sie $1^1/_2$ mal langsamer als die entsprechende Multiplikation.

(6) Potenzierung durch Multiplikation ersetzen.

Interrupt

Das CIA 1 löst alle $^1/_{60}$ Sekunden einen IRQ aus, um daraufhin den Programmzeiger in die Interruptroutine springen zu lassen. Hier wird die Tastatur abgefragt, die Zeit erhöht und noch einige andere Dinge erledigt. Die Bearbeitung und Beachtung des Interrupts zweigt natürlich einen Teil der ach so kostbaren Rechenzeit ab. Wenn wir den Interrupt verhindern, solange wir die Tastatur nicht benötigen, so kann auch mit dieser Technik ein Basic-Programm beschleunigt werden. POKE 56333,31 verhindert den IRQ, POKE 56333,159 gibt den IRQ wieder frei.

(7) Sperren des Interrupts.

Probieren Sie es aus. Verändern Sie Ihre alten Programme in der beschriebenen Weise und vergleichen Sie die Geschwindigkeit. Sie werden überrascht sein.

Zeitsparen einmal anders

Basic-Programme werden meist erst durch den Einsatz von Assemblerroutinen zu Programmen mit vernünftiger Laufgeschwindigkeit. Der Sprung in diese Routinen wird häufig durch einen SYS-Befehl realisiert. Die Parameterübergabe wird in diesen Fällen meist durch vorangehende POKE-Befehle bewerkstelligt. Sie sollen sich nun nicht mehr länger mit dieser umständlichen, uneleganten und vor allem langsamen Methode begnügen müssen. In unserem Computer verbirgt sich eine Funktion, die von vielen Handbüchern schlicht vergessen oder nur im Vorübergehen behandelt wird. Es handelt sich hierbei um die »USR-Funktion«. Im Gegensatz zum SYS-Befehl beinhaltet die USR-Funktion bereits eine vollständige Parameterübergabe. Sie kann also genauso benutzt werden wie alle anderen Funktionen des Basic-Interpreters (CHR$(x), ASC(x) ...). Der Vorteil dieser Funktion ist die freie Definition der eigentlichen Operation durch den Programmierer. Aber zunächst einmal die Form der Funktion: x1=USR(x2), wobei x2 der Wert ist, mit welchem die Routine arbeitet und x1 das Ergebnis der Operation darstellt. Die Parameter x1 und x2 können jede beliebige Form annehmen, wie zum Beispiel: Zahl, Variable, Zeichen, String (natürlich muß der Parameter zur jeweiligen Operation passen). x2 wird beim Aufruf der Routine automatisch in den FAC (Fließkommaakkumulator) übernommen. Analog dazu wird der Wert des FAC am Ende des Programmlaufs in x1 geschoben. Die weitere Funktionsweise der USR-Funktion ist ähnlich dem

```
0 REM **** STARTZEILE ****                              <099>
1000 INPUT A$:IF A$=""THEN END                          <197>
1010 FOR AA=0 TO LEN(A$)-1:POKE 513+AA,ASC
     (MID$(A$,AA+1,1)):NEXT:POKE 513+AA,0               <071>
1020 POKE 11,AA:SYS 830                                 <100>
```

Listing 5a. Basic-Teil einfügen

```
NAME : LISTING 5 MC     033E 037B
-----------------------------------
033E : A9 55 8D 02 03 A9 03 8D  DA
0346 : 03 03 A9 00 85 7A A9 02  0C
034E : 85 7B A9 30 4C 9C A4 A9  91
0356 : 83 8D 02 03 A9 A4 8D 03  7D
035E : 03 4C 71 A8 EA EA EA EA  80
0366 : EA EA EA EA EA EA EA EA  65
036E : EA EA EA EA EA EA EA EA  6D
0376 : EA EA EA EA EA EA EA EA  75
```

Listing 5b. Maschinen-Teil einfügen

```
NAME : LISTING 6 MC     033E 0353
-----------------------------------
033E : 20 6B E2 20 E2 BA 60 EA  2C
0346 : EA EA EA EA EA EA EA EA  45
034E : EA EA EA EA EA 9C A4 A9  3F
```

Listing 6. USR-Routine

```
NAME : LISTING 7 MC     033C 03F3
-----------------------------------
033C : 20 8D AD A6 2F A5 30 86  51
0344 : 5F 85 60 C5 32 D0 04 E4  BA
034C : 31 F0 1D A0 00 B1 5F C8  ED
0354 : C5 45 D0 06 A5 46 D1 5F  43
035C : F0 17 C8 B1 5F 18 65 5F  4B
0364 : AA C8 B1 5F 65 60 90 D7  16
036C : A2 E2 86 22 A9 03 4C 45  D4
0374 : A4 C8 B1 5F 18 65 5F 85  0A
037C : 24 C8 B1 5F 65 60 85 25  16
0384 : C8 B1 5F 20 96 B1 85 5F  CC
038C : 84 60 24 0E 30 1F 20 A2  CD
0394 : BB 18 90 04 20 67 B8 18  50
039C : A5 5F 69 05 85 5F 90 02  85
03A4 : E6 60 A4 60 C5 24 90 EC  89
03AC : C4 25 90 E8 60 20 D5 03  A8
03B4 : 20 0C BC 18 A5 5F 69 02  0B
03BC : 85 5F 90 02 E6 60 C5 24  26
03C4 : 90 06 A5 60 C5 25 B0 E4  DF
03CC : 20 D5 03 20 6F B8 4C B4  F3
03D4 : 03 A0 00 B1 5F AA C8 B1  2F
03DC : 5F A8 8A 4C 91 B3 41 52  1C
03E4 : 52 41 59 20 4E 4F 54 20  22
03EC : 46 4F 55 4E C4 00 00 00  45
```

Listing 7. Array-Berechnungen

```
NAME : LISTING 8 MC     033E 035D
-----------------------------------
033E : 20 FD AE 20 9E B7 8A 48  6F
0346 : 20 FD AE 20 9E B7 68 A8  AF
034E : 18 20 F0 FF 20 FD AE 4C  F8
0356 : A4 AA EA EA EA EA EA 17  47
```

Listing 8. Print AT-Simulation

SYS-Befehl beziehungsweise dem Aufruf einer normalen Basic-Funktion. Nach Aufruf der Funktion wird ein Assemblerprogramm angesprungen. Die Adresse dieses Programmes muß jedoch nicht bei jedem Aufruf erneut angegeben werden, sondern wird im sogenannten USR-Vektor (785/786) vordefiniert. Falls Sie die USR-Funktion bereits einmal ausprobieren wollten und als Antwort ein »SYNTAX ERROR« erhielten, so lassen Sie sich bitte davon nicht irritieren. Der USR-Vektor wird, wie alle anderen Vektoren, während des »Start-Resets« auf eine festgelegte Routine des Betriebssystems gelegt. Der Startwert des USR-Vektors zeigt dann auf eine Routine, die einen »SYNTAX ERROR« ausgibt. Aber nun zu unserer ersten Anwendung. Sie finden in Listing 6 eine Routine, die von dem angegebenen Wert den Sinuswert errechnet und diesen mit 10 multipliziert (Start an Adresse 830). Innerhalb von 100 Durchläufen erreicht unsere Routine einen Vorsprung von $^{18}/_{60}$ Millisekunden gegenüber einem entsprechenden Basic-Programm. Wie Sie an diesem Beispiel erkennen können, ermöglicht es die USR-Funktion, sowohl von Basic aus zugängliche Routinen wie auch Interpreter-interne Operationen aufzurufen und für eigene Zwecke zu nutzen. In Listing 7 haben wir für Sie bereits eine etwas anspruchsvollere Routine vobereitet. Dieses Programm berechnet aus einem beliebigen Variablenfeld (Array) die Summe der einzelnen Argumente. In diesem Fall muß bei Aufruf der USR-Funktion der Namen des Feldes angegeben werden und der USR-Vektor auf Adresse 828 gerichtet sein. Wenn Sie dieses Vorhaben von Basic aus realisieren möchten, müssen Sie schon eine Weile warten. Sie werden bereits bemerkt haben, daß Ihnen zur sinnvollen Nutzung der USR-Funktion nicht nur der Befehlssatz der 65xx-Prozessoren geläufig sein sollte. Der richtige Einsatz der ROM-Routinen ist mindestens genauso wichtig. Zu diesem Zweck ist die Benutzung eines ROM-Listings unumgänglich. Nachfolgend erwartet Sie eine kurze Aufstellung der wichtigsten Routinen. Bitte achten Sie bei der Benutzung der Arithmetik-Sequenzen auf die Einsprungadressen. Teilweise befindet sich zu Beginn der Routinen eine Abfrage auf FAC=0. Diese Abfrage sollte beim Aufruf über den USR-Vektor übergangen werden (Weitere ausführliche Informationen über Rechenoperationen und deren Anwendung finden Sie in dem Artikel »Rechnen in Maschinensprache«).

B849 : FAC = FAC+0.5
B850 : FAC = Konstante-FAC
B853 : FAC = ARG-FAC
B867 : FAC = FAC+ARG
BA28 : FAC = Konstante*FAC
BA2B : FAC = ARG*FAC
BA8C : ARG = Konstante
BAE2 : FAC = FAC*10
BB0F : FAC = Konstante/FAC
BB12 : FAC = ARG/FAC
BBA2 : FAC = Konstante
BBFC : FAC = ARG
BC0C : ARG = FAC
BF78 : FAC = ARG$^{\text{Konstante}}$
BF7B : FAC = ARG$^{\text{FAC}}$

Wie bereits erwähnt, können natürlich alle von Basic aus nutzbaren Routinen ebenfalls verwendet werden (SQR, SIN ...). Abschließend noch eine Anregung. Versuchen Sie einmal, mehrere Basic-Befehle unter der USR-Funktion zusammenzufassen. Auch durch diesen Trick kann man Basic beschleunigen.

SYS und mehr

Gerade haben wir behauptet, die USR-Funktion sei der Weisheit letzter Schluß, und jetzt beschäftigen wir uns doch wieder mit dem SYS-Befehl. Bitte glauben Sie nicht, wir wären inkonsequent. Es soll hier nur noch auf eine weitere Technik der Datenübergabe hingewiesen werden, die übrigens auch im Umgang mit der USR-Funktion verwendet werden kann. Auf einfachste Weise ist es möglich, nicht nur einen Parameter (USR), sondern beliebig viele zu übernehmen. Hierfür bieten sich gleich drei Routinen an:

AE83 : holt das nächste Element eines Ausdrucks in FAC
B79E : holt ein Byte in das X-Register
0073 : holt nächstes Zeichen in Akku

Das Betriebssystem hält sogar noch einige Routinen zur Prüfung der zu übergebenden Parameter bereit. Aber bitte bemühen Sie in diesen Fällen Ihr ROM-Listing. Zum Abschluß finden Sie in Listing 8 eine PRINT-AT-Simulation. Der Einsprung erfolgt über SYS 830, spalte, zeile, »text« (siehe auch »Rechnen in Maschinensprache«). (Erhart/do)