Memory Map mit Wandervorschlägen (Teil 17)

Diesmal kommen Speicherbereiche an die Reihe, die besonders für Maschinenprogrammierer interessant sind. Außerdem besprechen wir indirekte Sprung-Vektoren auf Basic-Routinen und Speicherzellen, die uns die Basic-Befehle SYS und USR näher bringen.

Die folgenden 4 Speicherzellen, nämlich 674 bis 678, werden nur vom C 64 benutzt. Beim VC 20 sind sie nicht belegt und können frei verwendet werden.

Adresse 674 ($2A2)

Indikator für das Steuerregister A des CIA #1

Mit CIA werden die beiden »Complex Interface Adapter« des C 64 bezeichnet. Das sind integrierte Schaltkreise, die Ein- und Ausgabeoperationen steuern. Jeder der beiden CIAs hat mehrere Register.

Das Steuerregister A (Adresse 56334 beziehungsweise $DC0E) beeinflußt die Zählregister des CIA, die ihrerseits die Ein- und Ausgabe von Daten auf beziehungsweise von Kassetten steuern. Das Betriebssystem speichert zu diesem Zweck geeignete Bitmuster in der Speicherzelle 674 ab, die von da in das Steuerregister transferiert werden.

Adresse 675 ($2A3)

Speicher für das Interrupt-Steuerregister B des CIA #1

Ein weiteres Register (Adresse 56333 beziehungsweise $DC0D) ist für die Unterbrechungen (Interrupt) des Computers bei Ein- und Ausgaben zuständig.

In der Speicherzelle 675 werden Werte dieses Interruptregisters beim Lesen von der Kassette zwischengespeichert.

Adresse 676 ($2A4)

Zusatzspeicher für Steuerregister B des CIA #1

Derselbe Wert, der bei der Vorbereitung des Lesevorganges von der Kassette in die Speicherzelle 674 kommt, gelangt auch nach 676, von wo er zu einem späteren Zeitpunkt beim Lesen zu Vergleichszwecken herangezogen wird.

Adresse 677 ($2A5)

Zwischenspeicher für das Link-Byte während des Bildschirm-Scrollens

Das Betriebssystem enthält eine Routine, welche den Bildschirminhalt hochschiebt (scrollt), sobald eine leere Zeile eingeschoben wird. Das bedeutet, daß jedesmal die Angaben in den Link-Tabellen der Speicherzellen 217 bis 241 geändert werden müssen. In der Speicherzelle 677 wird nun das Link-Byte zwischengespeichert, während der obere Teil des Bildschirms hochgeschoben wird.

Beim VC 20 gibt es diese Funktion übrigens auch. Sie wird durch die Speicherzelle 242 ausgefüllt.

Adresse 678 ($2A6)

Flagge für PAL oder NTSC

Im Gegensatz zum VC 20, der entweder fest auf die deutsche Fernsehnorm PAL oder aber auf die amerikanische Norm NTSC eingestellt ist, kann der C 64 beide Normen verkraften.

Diese beiden Normen beziehen sich unter anderem auf die Anzahl der Zeilen und auf die Abtast-Geschwindigkeit des Lichtstrahls im Fernsehgerät oder im Monitor.

Das Betriebssystem des C 64 überprüft gleich beim Einschalten, ob eine Rasterzeile 311 im angeschlossenen Sichtgerät vorhanden ist. Ist sie nicht vorhanden, muß alles auf die NTSC-Norm eingestellt werden, da diese nur 262 Rasterzeilen hat und mit einer internen Taktfrequenz von 14,3 MHz läuft.

Ist eine Rasterzeile 311 vorhanden, wird auf PAL-Norm eingestellt mit einer Taktfrequenz von 17,7 MHz.

Das Resultat dieses Tests wird in der Speicherzelle 678 gespeichert, und zwar als 0 für NTSC und 1 für PAL.

Adresse 679 bis 767 ($2A7 bis $2FF)

nicht belegt

Diese 89 Byte sind frei und können für alle möglichen Programme und Anwendungen verwendet werden. Beim VC 20 stehen sogar 95 Byte zur Verfügung, da der freie Bereich ja schon ab Speicherzelle 673 beginnt.

Dieser Speicherbereich hat den Vorteil, daß er — wie der Kassettenpuffer ja auch — von Basic nicht gestört wird. Er kann also für kleinere Maschinenprogramme oder auch für Sprite-Blöcke verwendet werden. Gegenüber dem Kassettenpuffer hat dieser Bereich den Vorteil, daß er durch Kassettenoperationen nicht gestört wird.

Die nächsten 12 Speicherzellen enthalten 6 Vektoren, deren Bedeutung bei der Übersetzung von Basic-Programmen im Texteinschub »Indirekte Sprung-Vektoren« näher erklärt wird.

Adresse 768 bis 769 ($300 bis $301)

Vektor auf die Ausgabe von Fehler-Meldungen (ERROR)

Dieser Vektor zeigt auf die Anfangsadresse der Basic-Routine, welche für die leidigen Fehlermeldungen zuständig ist. Beim C 64 zeigt der Vektor auf 58251 ($E38B), beim VC 20 auf 50234 ($C438).

Diese Routine verwendet eine Tabelle im Basic-Übersetzer, in der alle Fehlermeldungen gespeichert sind. Sie liegt im Speicherbereich 41374 bis 41767 (beim VC 20 49566 bis 49959). Die Routine verwendet den Inhalt des X-Registers (siehe Speicherzelle 781), um die entsprechende Fehlermeldung ganz einfach durch Abzählen der Reihenfolge aus der Tabelle auszulesen und auf dem Bildschirm anzuzeigen.

Ein Verbiegen dieses Vektors ist für zwei Anwendungsfälle sinnvoll.

Man kann die Fehlermeldung abschalten, um zu prüfen, ob ein bestimmtes Peripherie-Gerät, zum Beispiel das Floppylaufwerk, angeschlossen beziehungsweise eingeschaltet ist. Die Fehlermeldung ist abschaltbar mit POKE 768,61. Wieder eingeschaltet wird sie mit POKE 768,139. Ein Anwendungsbeispiel habe ich bereits in der Ausgabe 9/85, Seite 112 gebracht.

Die zweite Anwendung einer Verbiegung zielt auf eine Übersetzung der Fehlermeldungen. Wem der vorgegebene englische — und manchmal nicht gerade einleuchtende — Text der Fehlermeldungen nicht gefällt, kann den Vektor auf einen Speicherbereich legen, in dem er seine speziellen deutschen Fehlermeldungen abspeichert. Eine genaue Kenntnis der Fehlermeldungsroutine ist dazu allerdings erforderlich.

Adresse 770 bis 771 ($302 bis $303)

Vektor auf die Hauptroutine zur Ausführung von Basic-Befehlen

Dieser Vektor zeigt auf die Adresse 42115 ($A483), beim VC 20 auf 50307 ($C483). Die dort beginnende Routine steuert den Direkt-Modus, indem sie entweder direkt eingegebene Befehle ausführt oder mit Zeilennummer eingegebene Anweisungen abspeichert.

Adresse 772 bis 773 ($304 bis $305)

Vektor auf die Basic-Routine, die ASCII-Text in Token umwandelt

Dieser Vektor zeigt auf 42364 ($A57C), beim VC 20 auf 50556 ($C57C). Dort beginnt eine Routine, die nach dem Drücken der RETURN-Taste alle Anweisungen der damit eingegebenen Zeile absucht und Text beziehungsweise Wörter, die nicht zwischen Gänsefüßen stehen, als Basic-Befehle interpretiert und sie dann in sogenannte »Token« umwandelt. Token sind Codezahlen, die im Computer anstelle von Textbefehlen verwendet werden. Sie sind im Texteinschub »Die Kurzschrift von Basic« näher beschrieben.

Dieser Vektor kann verbogen werden, um zusätzliche Basic-Befehle zu erfinden und in das Betriebssystem einzubauen.

Adresse 774 bis 775 ($306 bis $307)

Vektor auf die Basic-Routine, die Token in ASCII-Werte zurückwandelt (LIST)

Dieser Vektor zeigt auf die Adresse 42778 ($A71A), beim VC 20 auf 50970 ($C71A). Dort beginnt eine Routine, die Token wieder in LISTbaren Text umwandelt. Sie steht nicht allein, sondern wird als Unterpro-

gramm von der LIST-Routine verwendet.

Falls ein Programmierer spezielle zusätzliche Basic-Befehle erfunden hat, kann er durch Verbiegen dieses Vektors seine eigenen Token lesbar ausLISTen.

Man kann auch durch eine entsprechende Verbiegung erreichen, daß die LIST-Routine nicht angesprungen werden kann, was gleichbedeutend ist mit einer LIST-Sperre. Das ist aber wohl nur sinnvoll bei einem Autostart-Programm.

Besser finde ich da ein kleines Programm, das J.Pellechi in der Zeitschrift RUN Ausgabe 6/85 (Seite 10) angegeben hat:

```
10 FOR J=679 TO 688
20 READ K
30 POKE J,K
40 NEXT J
50 POKE 774,167:POKE 775,2
60 NEW
70 DATA 72,173,141
80 DATA 2,208,251,104
90 DATA 76,26,167
```

Beim VC 20 ist nur die Zeile 90 verschieden:

```
90 DATA 76,26,199
```

In den freien Speicherbereich ab Speicherzelle 679 wird ein kleines Maschinenprogramm gePOKEt, das in den DATA-Zeilen 70 bis 90 steht. In Zeile 50 steht der für unser Beispiel entscheidende Befehl: Der Vektor in 774/775 wird nach der Adresse 679 verbogen. Dadurch springt die LIST-Routine immer zuerst auf die Adresse 679 in der sie das kleine Maschinenprogramm findet.

Disassembliert schaut das so aus:

```
02A7 48       PHA
02A8 AD 8D 02 LDA 028D
02AB D0 FB    BNE 02A8
02AD 68       PLA
02AE 4C 1A A7 JMP A71A
```

Zuerst wird der Akkumulator mit dem Inhalt der Speicherzelle 653 ($28D) geladen. Dort steht bekanntlich eine Zahl von 1 bis 7, je nachdem, ob die SHIFT-, CTRL- oder Commodore-Taste gedrückt ist. Ist dies der Fall, springt das Programm auf die Adresse 680 zurück und bildet so eine Dauerschleife, bis die Taste wieder losgelassen wird. Erst dann geht es weiter mit der ursprünglichen Zieladresse des Vektors in 774/775, nämlich $A71A (42778) beziehungsweise $C71A (50970) beim VC 20.

Auf diese Weise können Sie das LISTen eines Programms mit einer der drei genannten Tasten anhalten.

Adresse 776 bis 777 ($308 bis $309)

Vektor auf die Basic-Routine, die den nächsten Befehl liest und ausführt

Dieser Vektor zeigt auf die Adresse 42980 ($A7E4), beim VC 20 auf 51172 ($C7E4). Diese Routine prüft das nächste Token, ob es gültig ist. Wenn der ASCII-Wert des Token kleiner als 128 ist, wird er als Zeichen einer Variablen angesehen, und das System springt auf die LET-Routine. Das erklärt, warum zur Definition einer Variablen der LET-Befehl auch weggelassen werden kann.

Durch Verbiegen dieses Vektors kann zum Beispiel eine Trace-Routine gebaut werden, welche zuerst die Nummer der Zeile ausdruckt, die gerade ausgeführt wird, bevor sie auf die ursprüngliche Zieladresse des Vektors zurückkehrt.

Adresse 778 bis 779 ($30A bis $30B)

Vektor auf die Basic-Routine, die einen numerischen Ausdruck in eine Gleitkommazahl umwandelt

Dieser Vektor zeigt auf 44675 ($AE83), beim VC 20 auf 52867 ($CE83). Hier beginnt eine Routine, die einen einzelnen numerischen Wert, wenn er Teil eines Ausdrucks ist, von seinem ASCII-Wert in eine Gleitkomma-Zahl umwandelt.

Ist der Ausdruck eine Konstante, wird diese Umwandlung durchgeführt.

Ist der Ausdruck eine Variable, wird ihr Zahlenwert aus dem Variablenspeicher geholt.

Ist der Ausdruck die Zahl »pi«, wird der Zahlenwert für »pi« in den Gleitkomma-Akkumulator gebracht.

Der SYS-Befehl holt aus den nächsten vier Speicherzellen alle notwendigen Parameter, die für ein mit SYS zu startendes Maschinenprogramm notwendig sind. Er speichert sie in die vier Register des Mikroprozessors 6510 (beim VC 20 heißt er 6502). Es sind dies:

— der Akkumulator
— das X-Register
— das Y-Register
— das P-(Status-)Register

Die Bedeutung der Register ist im Assembler-Kurs erklärt worden.

Normalerweise funktioniert der SYS-Befehl nur, wenn vorher schon alle Parameter des aufgerufenen Maschinenprogramms richtig vorhanden sind, was meistens nicht der Fall ist.

So können Sie zum Beispiel mit Aufrufen der Load-Routine durch SYS 62622 nichts ausrichten, weil die für LOAD erforderlichen Parameter, nämlich Gerätenummer, File-Namen, Anfangs- und Endadresse, nicht festgelegt sind.

Wie dies mit Hilfe der vier folgenden Register-Speicherzellen erreichbar ist, hat Rolf Zweifel schon in der Ausgabe 7/84, Seite 131 erklärt. Weil das aber schon lange her ist und weil es hier so schön in den Kurs paßt, wiederhole ich dieses Thema im Texteinschub »Der vorbereitete SYS-Befehl«.

Adresse 780 ($30C)

Speicher für den Akkumulator

Adresse 781 ($30D)

Speicher für das X-Register

Adresse 782 ($30E)

Speicher für das Y-Register

Adresse 783 ($30F)

Speicher für das Statusregister

Die nächsten drei Speicherzellen 784 bis 786 sind beim VC 20 nicht belegt. Beim C 64 entsprechen sie den Adressen 0 bis 2 des VC 20.

Adresse 784 bis 786 ($310 bis $312)

Sprungbefehl und wählbare Sprungadresse des USR-Befehls

Mit dem Basic-Befehl USR wird bekanntlich ein Maschinenprogramm gestartet. Diese drei Speicherzellen werden bei der Abwicklung von USR verwendet. In ihnen muß der Anwender des USR-Befehls die Zieladresse in Low/High-Byte Darstellung angeben, ab der das Maschinenprogramm im Speicher steht.

Dieser Vorgang ist bereits im ersten Teil des Kurses in Ausgabe 11/84 behandelt worden bei den Speicherzellen 0 bis 2 des VC 20, die ja genau den Speicherzellen 784 bis 786 des C 64 entsprechen.

Da ich annehme, daß viele Leser dieses frühe Heft nicht besitzen, werde ich die Erklärung des USR-Befehls im Texteinschub »Das Mauerblümchen USR« in der nächsten Ausgabe wiederholen.

Adresse 787 ($313)

beim C 64 und VC 20 nicht belegt

Während dieses freie Byte des C 64 nicht viel nutzt, haben VC 20-Besitzer immerhin vier aufeinanderfolgende freie Bytes für eigene Vektoren und andere zwischenzuspeichernde Werte zur Verfügung, die nie in Gefahr geraten, von einem Basic-Programm überschrieben zu werden.

Das nächste Mal kommen wir mit den indirekten Sprung-Vektoren auf Routinen des Betriebssystems und mit dem Kassettenpuffer an das Ende dieses Kurses. (Dr.H.Hauck/ah)

Texteinschub #1 Indirekte Sprung-Vektoren

Mit »Vektor« wird ein Adressenpaar bezeichnet, dessen Inhalt in der Low/High-Byte-Darstellung wiederum eine Adresse angibt, ab der ein Maschinenprogramm beginnt.

Wenn man nun mit dem Maschinencode-Befehl JMP (jump) auf die Adresse springt, die der Vektor angibt, läuft das Maschinenprogramm ab dieser Adresse los.

Bekanntlich stehen im nicht veränderbaren ROM-Speicher viele Unterprogramme (Routinen) des Basic-Übersetzers und des Betriebssystems, die auch für andere Programme verwendbar sind. Commodore hat nun die brillante Idee gehabt, mehrere dieser Routinen herauszusuchen und ihre Anfangsadressen zur leichten Verwendung benutzerfreundlich in einer Tabelle zusammenzustellen, wo sie mit dem schon genannten Sprungbefehl »angewählt« werden können.

Diese Tabelle ist deshalb interessant, weil die Anfangsadressen der Routinen bei den einzelnen Commodore-Computern verschieden sind, obwohl sie eigentlich fast identische Übersetzer und Betriebssysteme haben. So beginnt zum Beispiel die LOAD-Routine des C 64 ab Adresse 62622 ($F49E), beim VC 20 aber ab Adresse 62786 ($F542).

Um zu erreichen, daß Programme, die diese Routinen benutzen, trotzdem von einem Commodore-Computer auf einen anderen übertragbar sind, hat Commodore diese Tabelle geschaffen, welche den Sprung auf diese Routinen unabhängig vom Computertyp macht.

Sie liegt (bei allen Commodore-Typen) im Bereich 768 bis 779 für Routinen des Basic-Übersetzers und 788 bis 819 für Routinen des Betriebssystems.

Diesen Zusammenhang zeige ich aber besser an einem Beispiel:
Der Vektor auf die LOAD-Routine hat die Adresse 816/817.
Wir schauen nach, was dort steht:

```
PRINT PEEK(816) PEEK(817)
```

Wir erhalten beim C 64 die Zahlen 158 und 244.
Beim VC 20 lautet das Ergebnis 66 und 245
Beide Zahlenpaare werden nach der üblichen Low/High-Byte Methode umgerechnet:
158 + 256 * 244 ergibt 62622.
66 + 256 * 245 ergibt 62786.

Das sind aber genau die weiter oben schon genannten Anfangsadressen der LOAD-Routine im Betriebssystem. Mit einem Sprung auf 816/817 landet ein Maschinenprogramm also immer zwangsläufig auf der LOAD-Routine. Kenner wissen, daß ein derartiger Sprung »indirekt« sein muß, also mit dem Code $6C, der nicht auf die angegebene Adresse, sondern auf die in ihr enthaltene Zieladresse springt.

Diese indirekte Sprungmethode hat außer der schon erwähnten Unabhängigkeit vom Computertyp noch einen weiteren Vorteil:

Da der Vektor, der auf die Zieladresse zeigt, im RAM-Speicher liegt, kann er verändert werden. Das bedeutet, daß dem Programmierer die Möglichkeit geboten wird, in ursprünglich »festgefrorene« Routinen des Übersetzers (Interpreter) und des Betriebssystems beliebige Änderungen und Varianten einzubauen. Ich will Ihnen das mit einem zwar nutzlosen, aber dennoch verblüffenden Beispiel zeigen. Bekanntlich meldet sich der Computer nach dem Befehl LOAD mit der Anweisung »PRESS PLAY ON TAPE«, weil der Vektor in 816/817 auf die Routine zeigt, die den LOAD-Befehl ausführt.

Jetzt verbiegen wir den Vektor so, daß er auf die SAVE-Routine zeigt. Diese Routine beginnt ab Speicherzelle ($E156) — beim VC 20 ab 57683 ($E153). Diese Adresse POKEn wir in Low/High-Byte Darstellung nach 816/817.

```
POKE 816,86:POKE 817,225 (VC 20:POKE 816,83)
```

Wenn Sie jetzt LOAD eingeben, meldet sich der Computer mit »PRESS RECORD AND PLAY ON TAPE«, der Anweisung für SAVE.

Weitere Beispiele dafür finden Sie im Kurs bei den entsprechenden Speicherzellen. Diese Art der Modifikation entspricht der oft genannten Wedge-Methode, die auch ich in diesem Kurs erwähnt habe, und zwar bei der Besprechung der Speicherzellen 115 bis 138 (Folge 9 in der Ausgabe 8/85). Aber der Einsatz der indirekten Sprung-Vektoren ist halt viel eleganter.

Texteinschub #2
Die Kurzschrift von Basic

Immer wenn Sie eine Anweisung auf den Bildschirm schreiben, sei es als Programmzeile, sei es als Direkt-Eingabe, wird sie in den Tastaturpuffer gebracht. Sobald die Eingabe mit der RETURN-Taste abgeschlossen wird, werden die Anweisungen dieser Zeile entweder in den Programmspeicher gebracht oder aber direkt ausgeführt. Beides geschieht allerdings nicht sofort, da der Computer ja bekanntlich intern nicht mit Buchstaben und Dezimalziffern, sondern mit besonderen Codezahlen arbeitet. Deshalb wird der Text einer Zeile zuerst in diese Codewerte umgewandelt und in eine besondere Reihenfolge gebracht. Ziffern, Zeichen und Texte, die innerhalb von Gänsefüßen stehen, werden mit ihrem ASCII-Code abgespeichert. Basic-Befehle und Basic-Funktionen werden in Zahlen umgewandelt. Diese heißen in der Fachsprache »TOKEN«. Der Befehl PRINT wird also nicht als Folge von 5 ASCII-Werten, sondern als einzelnes Byte mit dem Wert 153 gespeichert. Da meines Wissens die Liste der Token nur im Programmierhandbuch des VC 20 und im 64'er, Ausgabe 9/84 im Kurs von Christoph Sauer veröffentlicht sind, gebe ich in der weiter unten folgenden Tabelle alle Werte noch einmal an. Bemerkenswert ist, daß für GET# kein eigener Token vorhanden ist, dafür aber einer (203) für einen Befehl, den es in den Handbüchern nicht gibt, nämlich GO. Das erklärt übrigens, warum der Befehl GOTO auch in der Form GO TO geschrieben werden darf. Die Routine, welche Basic-Befehle ausführt, erkennt nämlich die beiden Token an und kombiniert sie miteinander.

Interessant ist auch, daß die Befehle TAB und SPC in ihren Token gleich die linke Klammer mit einschließen. Nach dieser Tabelle sieht eine gespeicherte Zeile so aus:

```
10 IF A=5THEN PRINT TAB(X)
Link 10 0 139 65 178 53 167 32 153 32 163 88 41 0
Zeilen-Nr. IF A = 5 THEN PRINT TAB(X)
```

Diese Zahlen können Sie selbst überprüfen, indem Sie diese Zeile eingeben und dann den Anfang des Programmspeichers sichtbar machen:

```
FOR J=1 TO 20:PRINT PEEK(2048)+J);:NEXT
```

Beim VC 20 müssen Sie den von der Speichererweiterung abhängigen Anfang des Programmspeichers einsetzen. Sie werden dieselbe Zahlenreihe wie oben erhalten.

Die Technik, in einem Programm direkt die Token anstelle von Basic-Befehlswörtern zu verwenden, bieten dem Programmierer in Maschinensprache eine gute Möglichkeit, Speicherplatz zu sparen. Das kann insbesondere bei großen Textprogrammen, wie zum Beispiel bei Adventure-Spielen, nützlich sein. Der Vollständigkeit halber muß ich noch erwähnen, daß die Token bei dem LIST-Befehl wieder in ihre ursprüngliche Textform zurückgewandelt werden. Die Vektoren für die Wandel- beziehungsweise Rückwandel-Routinen stehen in 772/773 und 774/775.

128	END	147	LOAD	166	SPC(	185	POS
129	FOR	148	SAVE	167	THEN	186	SQR
130	NEXT	149	VERIFY	168	NOT	187	RND
131	DATA	150	DEF	169	STEP	188	LOG
132	INPUT#	151	POKE	170	+	189	EXP
133	INPUT	152	PRINT#	171	—	190	COS
134	DIM	153	PRINT	172	*	191	SIN
135	READ	154	CONT	173	/	192	TAN
136	LET	155	LIST	174	↑	193	ATN
137	GOTO	156	CLR	175	AND	194	PEEK
138	RUN	157	CMD	176	OR	195	LEN
139	IF	158	SYS	177	größer	196	STR$
140	RESTORE	159	OPEN	178	=	197	VAL
141	GOSUB	160	CLOSE	179	kleiner	198	ASC
142	RETURN	161	GET	180	SGN	199	CHR$
143	REM	162	NEW	181	INT	200	LEFT$
144	STOP	163	TAB(	182	ABS	201	RIGHT$
145	ON	164	TO	183	USR	202	MID$
146	WAIT	165	FN	184	FRE	203	GO

Tabelle der Token und deren Werte

Texteinschub #3
Der vorbereitete SYS-Befehl

Programme, die in Maschinensprache geschrieben sind, können von einem Basic-Programm aus mit dem SYS-Befehl angewählt und ausgeführt werden.

Im Prinzip gilt das auch für Routinen des Basic-Übersetzers (Interpreter) und des Betriebssystems, die fest im ROM-Speicher untergebracht sind.

Ein Beispiel dafür ist SYS 58260, der Sprung auf den Kaltstart — beim VC 20 ist es SYS 58232, der den Computer in die Ausgangslage zurücksetzt.

Die meisten dieser Routinen benötigen jedoch verschiedene Angaben — man nennt sie auch Parameter — die vor der Ausführung des SYS-Befehls richtig eingestellt sein müssen.

Die LOAD-Routine zum Beispiel, die ab Speicherzelle 62622 ($F49E) — beim VC 20 ab 62793 ($F549) — beginnt, können wir mit dem SYS 62622 nicht starten. Es fehlen die Angaben über Geräte-Nummer (8 für Floppy, 0 für Band), File-Namen, sowie Anfangs- und Endadresse. Diese Parameter werden normalerweise nach dem Befehl LOAD von der Routine des Interpreters, die den LOAD-Befehl übersetzt, eingegeben. Wir geben ja nicht einfach LOAD ein, wenn wir ein Programm mit dem Namen »Test« auf Diskette speichern wollen, sondern wir schreiben LOAD "TEST",8.

Auch wenn wir nur LOAD eintippen, werden vom Übersetzer Parameter gesetzt, nämlich »namenlos« und 0 für Bandgerät. Ich hoffe, Ihnen ist geläufig, daß beim Weglassen aller Angaben der Übersetzer immer Kassettenoperationen durchführt. Natürlich können wir uns das anschauen:

Die Routine des Übersetzers für den Basic-Befehl LOAD beginnt an Speicherzelle 57704 ($E168), beim VC 20 bei 57700 ($E164).

Mit SYS 57704 springen wir dorthin — und in der Tat, wir erhalten »PRESS PLAY ON TAPE«. Aber ein Programm auf diese Weise von der Floppy zu LOADen, gelingt uns nicht, es sei denn, wir können die fehlenden Parameter von Hand eingeben.

Genau das aber können wir, weil der SYS-Befehl sich diese Parameter aus den Speicherzellen 780 bis 783 holt und in die vier Register des Mikroprozessors schreibt.
780 ist die Adresse des Akkumulators
781 ist die Adresse des X-Registers
782 ist die Adresse des Y-Registers
783 ist die Adresse des Status-Registers.

Die Behandlung von A, X und Y ist unkompliziert, wie wir gleich sehen werden.

Das Status-Register, manchmal auch P-Register genannt, ist nicht so einfach zu verwenden, da es nicht Zahlenwerte, sondern Flaggen (Bitmuster) enthält. Im einzelnen bedeuten:

BIT Nr.	WERT	FLAGGE	ABKÜRZUNG
0	1	Übertrag	C(arry)
1	2	NULL	Z(ero)
2	4	Unterbrechung	I(nterrupt)
3	8	Dezimal	D
4	16	Abbruch	B(reak)
5	32	nicht benutzt	
6	64	Überlauf	V
7	128	Vorzeichen	N(egativ)

Um eine der Flaggen des Status-Registers zu löschen, empfiehlt es sich, das ganze Register mit POKE 783,0 zu löschen. Umgekehrt muß man beim Setzen der Bits sehr aufpassen wegen der Unterbrechungsflagge I. Eine 1 in I entspricht dem Maschinen-Befehl SEI, der alle Interrupts ausschaltet, auch die der Tastatur-Abfrage, was natürlich sehr störend sein kann! Um alle Flaggen außer der Unterbrechungsflagge I zu setzen, muß POKE 783,247 eingegeben werden.

So, jetzt wird es Zeit für ein Beispiel, wie vor dem SYS-Befehl Parameter eingegeben werden können. In der Literatur wird immer das Beispiel gewählt, den Cursor auf eine bestimmte Position zu setzen, beziehungsweise seine Position abzufragen. Dazu gibt es eine Routine, die bei beiden Computern ab Speicherzelle 65520 ($FFF0) beginnt.

Sie nimmt die Zahl, die im X-Register steht, und verwendet sie als Zeilennummer; die Zahl des Y-Registers nimmt sie als Spaltennummer, setzt dann den Cursor an diese Stelle und bringt die beiden Werte in die Speicherzellen 209/210 und 211.

Unser Beispiel hat die Aufgabe, den Cursor in die vierte Spalte der siebten Zeile zu setzen, dort das Dollar-Zeichen hinzuschreiben und es rot zu färben.

```
5 PRINT CHR$(147)
10 POKE 783,0
20 POKE 781,6
30 POKE 782,3
40 SYS 65520
```

Nach Löschen des Bildschirms werden zuerst alle Flaggen des Statusregisters gelöscht (Zeile 5). Dann kommt die Zeilennummer in das X-Register (Zeile 10) und die Spaltennummer in das Y-Register (Zeile 30). Nach dem Eingeben dieser Parameter können wir mit SYS auf die Routine springen.

```
50 ZEILE=PEEK(209)+256*PEEK(210)
60 ADRESSE = ZEILE + PEEK(211)
70 POKE ADRESSE,36
```

In Speicherzellen 209/210 können wir jetzt (zur Übung) die Zeilennummer ablesen. Die Adresse der Cursorposition im Bildschirmspeicher erhalten wir durch die Addition der Zeilennummer mit dem Inhalt der Speicherzelle 211. Dorthin POKEn wir den Bildschirmcode des Dollarzeichens, nämlich 36 (Zeile 70).

```
80 SYS 59940
90 FARBE=PEEK(243)+256*PEEK(244)
100 POKE FARBE+PEEK(211),2
```

Für das Färben des Dollarzeichens verwenden wir eine weitere Routine des Betriebssystems, die ab 59940 — beim VC 20 ab 60082 — beginnt. Sie ermittelt die Zeilenposition des Cursors im Farbspeicher und bringt diesen Wert in die Speicherzellen 243/244, wo wir ihn abfragen können (Zeile 90). Die Adresse der Cursorposition im Farbspeicher setzt sich aus diesem Wert plus der Spaltennummer zusammen, die wir wieder der Speicherzelle 211 entnehmen. Auf diesen Platz POKEn wir den Farbcode 2 für rot (Zeile 100). So leicht ist das, wenn man die Routinen und die Aufgaben der Speicherzellen kennt.

Die letzteren lernen Sie in diesem Kurs. Die Beschreibung und Anwendung der Routinen muß, wie schon öfters erwähnt, einem eigenen Kurs vorbehalten bleiben.