Computer persönlich

DAS VC-20 BUCH

Eine umfassende Information und Utilitiessammlung mit zahlreichen Programmen

Michael Hegenbarth/Michael Schäfer

Das VC-20 Buch

Michael Hegenbarth
Michael Schäfer

Das VC-20 Buch

Eine umfassende Information und Utilitiessammlung mit zahlreichen Programmen

Verlag Markt & Technik

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Hegenbarth, Michael:
Das VC 20 Buch : e. umfassende Information
u. Utilitiessammlung mit zahlr. Programmen /
Michael Hegenbarth ; Michael Schäfer. —
Haar bei München : Verlag Markt u. Technik, 1983.
(Computer persönlich)
ISBN 3-922120-50-4
NE: Schäfer, Michael:

ISBN 3-922120-50-4

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Schoder-Druck, 8906 Gersthofen
Printed in Germany

Vorwort

Der VC-20 hat sich nicht nur in der Bundesrepublik Deutschland sondern in der ganzen Welt als ein zumindest beachtenswerter Heim-Computer einen Namen gemacht. Dies beweist nicht nur die hohe Verkaufszahl (in den USA ca. 1.000.000, in der Bundesrepublik Deutschland ca. 100.000), sondern auch seine Qualität, welche man für einen recht günstigen Preis erwerben kann. Es wäre müßig, seine mannigfaltigen Möglichkeiten alle aufzuführen. Denn eingeweihten VC-20-Fans sind sie ohnehin geläufig.

Trotz der doch eigentlich nachgewiesenermaßen großen Akzeptanz des VC-20 muß der Benutzer eines solchen Rechners, und erst recht der Programmierer, feststellen, daß es mit der VC-20-Literatur ein wenig hapert, nicht nur in der BRD. Zwar kann man von VC-20-Programmen heute überschüttet werden, sowohl von den Spezialgeschäften als auch von zahlreichen Tausch-Vereinen oder Freunden/Bekannten, jedoch geben diese demjenigen, der manche Besonderheiten des VC-20 besser verstehen möchte, keine oder nur sehr spärliche Antworten auf viele anstehenden Fragen.

Dieses Buch ist also insbesondere auf diejenigen Leser zugeschnitten, die ebenfalls wie wir, die Autoren, die Absicht verfolgen, aus dem relativ harmlos aussehenden VC-20 möglichst viele seiner verborgenen Leistungen aufzuspüren und sinnvoll zu benutzen. Natürlich wird auch der VC-20-Anfänger mit, so hoffen wir, den meisten in diesem Buch enthaltenen Programmen direkt von Anfang an nutzbringende Eigenanwendungen realisieren (lassen) konnen. Jedoch setzen wir bewußt den Hebel beim eingefleischten VC-20-Benutzer/Programmierer an, weil wir glauben, daß die ersten aufkommenden Fragen ohnehin durch die bestehenden Bücher, z.B. das VC-20-Handbuch und VC-20-Programmier-Handbuch beantwortet werden.

Um es offen auszusprechen, dieses Buch nimmt also keinen Ausschließlichkeitscharakter für sich in Anspruch. "Zuarbeit" bzw. zusätzliche Information z.B. durch oben genannte Bücher sind zwar nicht unbedingt nötig, würden wir jedoch empfehlen.

Dieses Buch stellt zahlreiche Programmierhilfsmittel, Programme bzw. Programmmodule und Tricks zur Verfügung, die dem Leser nicht nur explizit seiner Lösungssuche für individuelle Programme, son-

dern auch implizit hilfreich zur Seite stehen können. Damit der Leser sich nicht mit der Eintipperei der zahlreichen Programme abquälen muß, bietet der Verlag dieses Buches den Versand der Kassette bzw. Diskette an, die alle Programme enthält.

Wie es nun mal bei Büchern so vorkommt, werden Sie, verehrte Leser, sicherlich auch auf Fehler stoßen. Sie lassen sich einfach nicht vermeiden. Ein Philologe, der bei der Wahl seiner Worte nicht so sehr auf das 6. Bit vom 10. Byte eines Programms achten muß, hat es bei seiner Schriftstellerei erheblich einfacher (selbstverständlich soll diese Bemerkung in keinster Weise den Wert eines solchen Buches in Frage stellen). Die Leser mögen also für solche von uns übersehenen Fehler Verständnis haben. Jeden von Ihnen kommender Kommentar bzw. Korrekturhinweis werden wir uns natürlich zu Herzen nehmen und in Folgeauflagen (falls Nachfrage besteht) berücksichtigen.

Mit diesem Buch verfolgt der Verlag ein neues Prinzip, woran wir in der ersten Zeit der Erstellung zu "knabbern" hatten. Der Verlag stellte uns einen komfortablen Textverarbeitungs-Computer zur Verfügung, in den wir gewissermaßen alles selbst eintippen mußten. Ein paar Wochen Verzögerung bei der Herausgabe ließ sich deswegen nicht vermeiden.

Wir danken dem Verlag dafür, daß er uns Buch-Pionieren die Möglichkeit gegeben hat, Ihnen, verehrte Leser, unserer (wir meinen kompetenter) Meinung nach hochwertige Programme (die natürlich gut getestet und vor allem selbst benutzt worden sind und werden) zugänglich zu machen.

Allerdings gilt unser Dank noch mehr unseren beiden Familien, die rührendes Verständnis (zwar oft mit verärgerter Anti-Computer-Mine) für dieses von uns Familienvätern momentan bevorzugte Hobby und der damit verbundenen Erstellung dieses Buches hatten.

Michael Hegenbarth und Michael Schäfer

INHALTSVERZEICHNIS

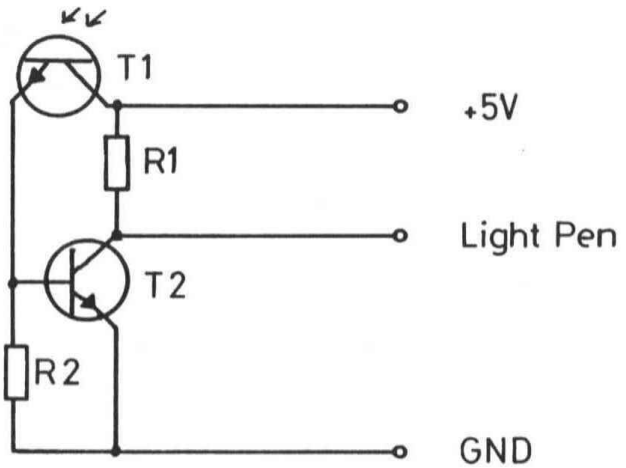
1

Nutzung von Peripheriegeräten

1.1 DER LICHTGRIFFEL

Der Lichtgriffel, auch Lightpen genannt, bietet dem Benutzer eine völlig neue Art, mit seinem Computer zu arbeiten. Man ist nicht mehr nur auf die Tastatur als Eingabegerät angewiesen, sondern kann seine Eingaben direkt auf dem Bildschirm machen.

Die Bezeichnungen Lichtgriffel oder Lightpen sind eigentlich falsch, denn mit dem Lichtgriffel kann man gar nicht schreiben. Die Spitze des Lichtgriffels besteht aus einem Foto-Transistor, der den Anodenstrahl der Bildrohre abfragt und an seinem Ausgang immer dann einen negativen Impuls liefert, wenn der Anodenstrahl auf den Foto-Transistor trifft. Dieser Impuls ist der Anreiz für den VIC, die aktuelle Position des Anodenstrahls als horizontale und vertikale Koordinaten in die beiden Kontrollregister CR6 und CR7 zu schreiben.



T1 = CQY 78 o.ä.
T2 = BC 107 o.ä.
R1 = 560Ω
R2 = 22 KΩ

Bild 1.1.1: Light Pen

1.1 Lichtgriffel

In der in Bild 1.1.1 gezeigten Schaltung liefert der Foto-Transistor T1 einen positiven Impuls, der mit T2 invertiert und verstärkt wird. Der Widerstand R2 hat die Aufgabe, T2 negativ vorzuspannen und so zu verhindern, daß bei normaler Raumbeleuchtung schon ein Impuls geliefert wird. Der Widerstand R1 ist der Lastwiderstand für T2 und hält den Ausgang des Lichtgriffels auf High-Potential. Der Nachbau des Lichtgriffels sollte auch dem ungeübten Bastler keine Schwierigkeiten bereiten, lediglich auf einen möglichst kompakten Aufbau sollte man achten, damit die Schaltung später in einem alten Kugelschreiber- oder Filzstiftgehäuse Platz findet. Der Anschluß an den Spiele-Port geschieht dann über ein dreiadriges Kabel.

BEDIENUNGSHINWEISE

Das hier behandelte Programm ist ein Beispiel für die Handhabung des Lichtgriffels. Nach RUN sehen Sie auf dem Bildschirm symbolische Schalter für die vier Tongeneratoren des VC-20. Durch Bestreichen dieser "Schalter" mit dem Lichtgriffel werden die Tongeneratoren eingeschaltet und in ihrer Frequenz geändert, solange sie den Schalter berühren. Das Basic-UP ab Zeile 60000 ist universell verwendbar und liefert mit den Variablen X und Y die Position des Lichtgriffels auf dem Bildschirm.

PROGRAMM-EINZELHEITEN

1.) Name: LICHTGRIFFELDEMO

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Lichtgriffelabfrage-Programm

4.) Anzahl der Bytes: 775

5.) Benutzte Variablen:

T1 - T4	Adressen der Tongeneratoren 1 bis 4
L	Adresse des Kontrollregisters für die Lautstärke
F1 - F4	Frequenz für die Tongeneratoren 1 bis 4
J	Schleifenzähler

X,Y — Koordinaten des Lichtgriffels auf dem Bildschirm:
X = 0 bis 21 (Spalten)
Y = 0 bis 22 (Zeilen)

6.) Listing: s. Bild 1.1.2

```
0 REM LICHTGRIFFELDEMO
10 T1=36874:T2=36875:T3=36876:T4=36877:L=36878
20 F1=127:F2=127:F3=127:F4=127
30 PRINT"■";
40 FORJ=1TO5
50 PRINT"■■■■■■■■■■■■■┌─┐":REM CRSR DWN + 10*CRSR RI
60 PRINT"■■■■■■■■■■■■■│o│":REM 10*CRSR RI
70 PRINT"■■■■■■■■■■■■■└─┘":REM 10*CRSR RI
80 NEXT
90 PRINT"■";:REM CRSR HOME
100 FORJ=1TO4:PRINT"■■   TON"J"■":NEXT:REM CRSR DWN
110 PRINT"■■    AUS":REM CRSR DWN
120 POKE L,15:POKET1,0:POKET2,0:POKET3,0:POKET4,0
130 GOSUB60000
140 IFY>0ANDY<4ANDX>9ANDX<13THENF1=F1+1:POKET1,F1:IFF1=255THENF1=127
150 IFY>4ANDY<8ANDX>9ANDX<13THENF2=F2+1:POKET2,F2:IFF2=255THENF2=127
160 IFY>8ANDY<12ANDX>9ANDX<13THENF3=F3+1:POKET3,F3:IFF3=255THENF3=127
170 IFY>12ANDY<16ANDX>9ANDX<13THENF4=F4+1:POKET4,F4:IFF4=255THENF4=127
180 IFY>16ANDY<20ANDX>9ANDX<13THEN120
190 GOTO130
60000 REM POSITION DES LIGHT-PEN NACH X U. Y
60010 Y=INT((PEEK(36871)-30)/4):IFY<0THENY=0
60020 IFY>22THENY=22
60030 X=INT((PEEK(36870)-47)/4):IFX<0THENX=0
60040 IFX>21THENX=21
60050 RETURN
```

Bild 1.1.2

7.) Erläuterungen zum Programm:

Zeilen 10 - 20: Die benutzten Variablen werden initialisiert

Zeilen 30 - 80:
Der Bildschirm wird geloscht, und eine Bildschirmmaske wird ausgegeben.

Zeilen 90 - 110:
Die Beschriftung der Schalter wird ausgegeben.

Zeile 120:
Die vier Tongeneratoren werden ausgeschaltet und vorbereitend die große Lautstarke eingeschaltet.

Zeile 130:
Sprung in die Subroutine; nach Ruckkehr enthalten X und Y die Koordinaten des Lichtgriffels.

Zeile 140:
Stimmen die Koordinaten des Lichtgriffels mit den Koordinaten des 1. "Schalters" überein? Wenn ja, dann wird die Frequenz des Tongenerators 1 um 1 erhöht und in das Kontrollregister CRA des VIC geschrieben.

Zeile 150: 2. "Schalter" ?

Zeile 160: 3. "Schalter" ?

Zeile 170: 4. "Schalter" ?

Zeile 180:
Aus-"Schalter" ? Wenn ja, dann werden die vier Tongeneratoren ausgeschaltet (Zeile 120).

Zeile 190: Neue Koordinaten holen

Das Basic-UP ab Zeile 60000 errechnet die Position des Lichtgriffels auf dem Bildschirm aus den Inhalten der beiden Kontrollregister CR6 und CR7 des VIC und schreibt sie in die Variablen X und Y.

CR6 enthält den Wert #30, wenn der Lichtgriffel auf die Oberkante des Bilschirmfensters zeigt, und den Wert #122, wenn er auf die Unterkante des Bildschirmfensters weist. Daraus ergeben sich 92 mögliche Vertikalpositionen.

CR7 enthält den Wert #47, wenn der Lichtgriffel auf die linke Kante des Bildschirmfensters zeigt, und den Wert #135, wenn er auf die rechte Kante des Bildschirmfensters weist. Daraus ergeben sich 88 mögliche Horizontalpositionen.

Zeile 60010:
Die Vertikalposition des Lichtgriffels auf dem Bildschirm wird in Zeilennummern umgerechnet und der Variablen Y übergeben. Die Zeilennummern 0 bis 22 sind möglich. Ist die Zeilennummer kleiner als 0, so wird sie auf 0 festgelegt.

Zeile 60020:
Ist die Zeilennummer größer 22, so wird sie auf 22 festgelegt

Zeile 60030:
Die Horizontalposition des Lichtgriffels auf dem Bildschirm wird in Spaltennummern umgerechnet und der Variablen X übergeben. Die Spaltennummern 0 bis 21 sind möglich.

Zeile 60040:
Die Spaltennummer wird auf ein Maximum von 21 begrenzt.

Zeile 60050: Rücksprung ins Hauptprogramm

1.2 DREHREGLER AM VC-20

Die beiden im Video-Interface-Chip (6561) integrierten A/D-Wandler erlauben unter anderem auch den Anschluß von zwei Drehreglern an den VC-20. Ein Drehregler, auch Paddle genannt, besteht aus einem regelbaren Widerstand der zwischen +5 V und dem Eingang des A/D-Wandlers (Pot X u. Pot Y) geschaltet wird (s. Bild 1.2.1).

1.2 Drehregler am VC-20

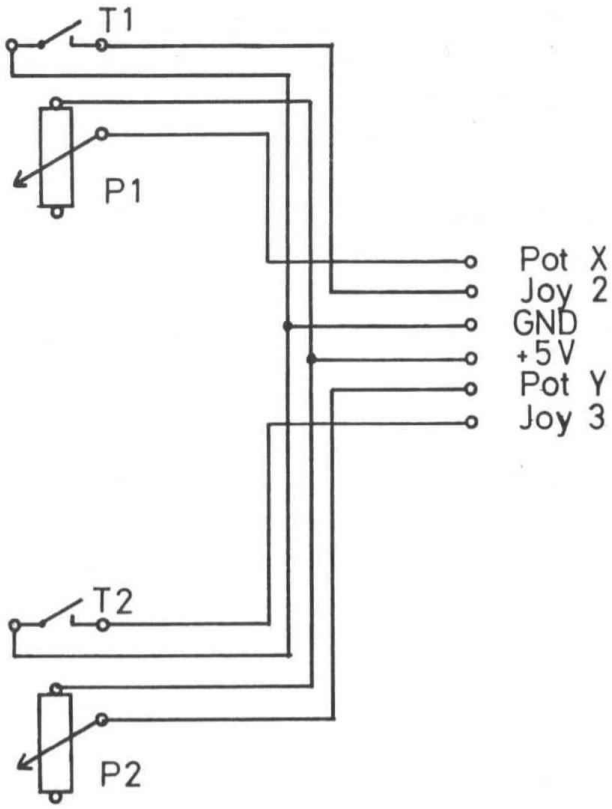


Bild 1.2.1: Paddle (Paar)

Je nach eingestelltem Widerstandswert liegt an diesem Eingang eine Spannung zwischen 0 V und +5 V an. Der A/D-Wandler konvertiert dieses analoge Signal in eine vom Prozessor lesbare 8-Bit-Zahl und stellt diese an den Kontrollregistern (CR8 und CR9) des VIC zur Verfügung. Die Adressen der Kontrollregister sind:

CR8 :	DEZ 36872	=	HEX 9008
CR9 :	DEZ 36873	=	HEX 9009

Die Inhalte der Kontrollregister können mittels PEEK(X) ausgelesen und Variablen zugewiesen werden.
Hier soll ein universell verwendbares Basic-Unterprogramm zur Drehreglerabfrage vorgestellt werden.

BEDIENUNGSHINWEISE

Das Basic-UP wird per GOSUB 60000 aufgerufen. Nach Ruckkehr aus dem UP enthalten die Variablen X und Y den digitalisierten Spannungswert vom Pot X bzw. Pot Y. Die Variablen FX und FY enthalten 1, wenn die zu den Drehreglern gehorenden Feuerknopfe gedruckt sind.

PROGRAMM-EINZELHEITEN

1.) Name: DREHREGLER

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Drehreglerabfrage-Unterprogramm

4.) Anzahl der Bytes: 256

5.) Benutzte Variablen:

X	Inhalt vom CR8 (Pot X)
Y	Inhalt vom CR9 (Pot Y)
FX	Feuerknopf Paddle X
FY	Feuerknopf Paddle Y

6.) Listing: s. Bild 1.2.2

```
0 REM PADDLE
10 GOSUB60000:PRINTX;Y,FX;FY:RUN
60000 X=PEEK(36872): REM POT X
60010 Y=PEEK(36873): REM POT Y
60020 POKE37139,239: REM VIA#1 PA4=EINGANG
60030 POKE37154,127: REM VIA#2 PB7=EINGANG
60040 FY=0
60050 FX=0
60060 IFNOTPEEK(37137)AND16THENFX=1
60070 IFNOTPEEK(37152)AND128THENFY=1
60080 POKE37154,255:POKE37139,128
60090 RETURN
```

Bild 1.2.2

7.) Erläuterungen zum UP:

Zeile 60000: Variable X übernimmt den Inhalt von CR8 (Pot X)

Zeile 60010: Variable Y übernimmt den Inhalt von CR9 (Pot Y)

Zeile 60020:
Der Port PA4 des VIA#1 wird im Datenrichtungsregister als Eingang gekennzeichnet (Joy 2).

Zeile 60030:
Der Port PB7 des VIA#2 wird im Datenrichtungsregister als Eingang gekennzeichnet (Joy 3).

Zeile 60040: Feuerknopf Y nicht gedrückt

Zeile 60050: Feuerknopf X nicht gedrückt

Zeile 60060:
Wenn Bit 4 des Eingaberegisters vom Port A logisch 0 ist, dann ist der Feuerknopf gedrückt und FX=1.

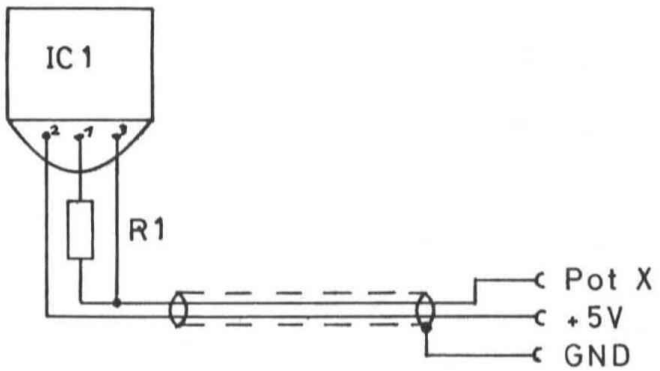
Zeile 60070:
Wenn Bit 7 des Eingaberegisters vom Port B logisch 0 ist, dann ist der Feuerknopf gedrückt und FY=1.

Zeile 60080:
Hier werden die Datenrichtungsregister zurückgesetzt. Würde dies nicht geschehen, wären einige Tasten der Tastatur gesperrt und ein Arbeiten mit dem Diskettenlaufwerk wäre nicht mehr möglich.

Zeile 60090: Rücksprung aus dem Unterprogramm

1.3 VC-20 ALS THERMOMETER

Eine weitere Moglichkeit, einen der beiden im VIC integrierten A/D-Wandler zu nutzen, soll in diesem Kapitel beschrieben werden. Anstelle eines Potentiometers wird in diesem Beispiel ein temperaturabhangiger Widerstand zwischen +5 V und dem Eingang des A/D-Wandlers geschaltet. In unserem Beispiel wird der integrierte Konstantstromregler LM334 als Temperaturfühler (s. Bild 1.3.1) benutzt. Der Aufwand für Hardware ist gering und das PGM recht einfach.



IC 1 = LM 334
R1 = 1 KΩ

Der Widerstand wird direkt am Temperaturfühler angelötet.
Unbedingt abgeschirmte Zuleitung verwenden.

Bild 1.3.1: Temperaturfuhler

PROGRAMM-EINZELHEITEN

1.) Name: THERMOMETER

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Temperaturanzeige-Programm

1.3 VC-Thermometer

4.) Anzahl der Bytes: 232

5.) Benutzte Variablen:

CH$	CLR HOME
CD$	CRSR DWN
CU$	CRSR UP
HO$	HOME
BL$	7 * SPACE
P1$	CH$ + CD$
P2$	HO$ + CD$ + BL$ + CU$
T	Temperatur in Celsius

6.) Listing: s. Bild 1.3.2

```
0 REM THERMOMETER
10 CH$="■":REM CLR HOME
20 CD$="■":REM CRSR DWN
30 CU$="■":REM CRSR UP
40 HO$="■":REM HOME
50 BL$="       ":REM 7*SPACE
60 P1$=CH$+CD$
70 P2$=HO$+CD$+BL$+CU$
80 PRINTP1$;SPC(9)"GRAD CELSIUS"
90 T=(255-PEEK(36872)-70)*.5
100 PRINTP2$:PRINTT
110 GOTO90
```

Bild 1.3.2

7.) Erlauterungen zum PGM:

Die in unserem Beispiel (Bild 1.3.1) dimensionierte Schaltung erlaubt es, Temperaturen im Bereich -35 bis +92.5 Grad Celsius zu messen. Bei 0 Grad liegt an Pot X (Kontrollregister CR8, Adresse $9008) eine Spannung von 2.1 V an, was einem Wert von #185 in CR8 entspricht. Eine Temperaturanderung um +1 Grad andert den in CR8 ablesbaren Wert um -2.

Zeilen 10 - 70:
Die Variablen zur Bildschirmausgabe werden initialisiert.

Zeile 80: Die Bildschirmmaske wird ausgegeben.

Zeile 90:
CR8 wird ausgelesen, die Temperatur errechnet und der Variablen T übergeben.

Zeile 100:
Die Temperatur wird in die Bildschirmmaske geschrieben.

1.4 FUNKFERNSCHREIBEN MIT DEM VC-20

Die Betriebsart Funkfernschreiben (RTTY) erfreut sich bei den Funkamateuren großer Beliebtheit. Wenn vor ein paar Jahren noch die von der Post ausrangierten mechanischen Fernschreiber bei den Amateuren hoch im Kurs standen, so sind doch zur Zeit schon sehr viele Computer im Einsatz. Angefangen von reinen Nachrichten-Computern, die nur Funkfernschreibsignale (ASCII oder Baudot) und Morse-Zeichen empfangen bzw. senden können, bis hin zu Home-Computern wie dem VC-20.

Wegen der großen Datenmenge, die in kürzester Zeit anfällt, sind RTTY-PGMe für Home-Computer zumeist in Maschinensprache geschrieben. Das hier vorgestellte PGM jedoch ist nur in Basic geschrieben und nutzt die im VC-20 implementierte V.24 (RS232)-Schnittstelle aus. Diese Schnittstelle ist interrupt-gesteuert und arbeitet unabhängig vom Basic-PGM, so daß genügend Zeit bleibt, die empfangenen Zeichen vom Baudot- in den ASCII-Code umzuwandeln und auf dem Bildschirm darzustellen, bzw. die Tastatur abzufragen und die eingegebenen Zeichen in den Baudot-Code umzuwandeln und auszusenden.

BEDIENUNGSHINWEISE

Das PGM gliedert sich in 3 Teile:

1. Allgemeiner Teil (Zeilen 10-210)
   - Festlegen der Baud-Rate
   - öffnen des Schreib- und Lesekanals
   - Eingabe der Fix-Texte
   - Umwandlungstabelle Baudot --> ASCII lesen
   - Umwandlungstabelle ASCII --> Baudot lesen

2. Empfang
   - Kanal #2 lesen, prüfen, umwandeln und Zeichen auf dem Bildschirm darstellen
   - Tastatur abfragen, prüfen und Befehl ausführen

3. Sendung
   - Tastatur abfragen, prüfen, umwandeln und Zeichen in Kanal #1 schreiben
   - bzw. Befehl ausführen

Belegung der Tastatur mit Sonderfunktionen:

- Bei Sendung und Empfang:

F1	Sende/Empfangsumschaltung
CRS DWN	= Bu-Taste
CRS RI	= Zi-Taste

- Nur bei Empfang:

0 - 6	Baud-Rate ändern
0	45.5 Baud
1	50 Baud
2	75 Baud
3	110 Baud
4	134.5 Baud
5	150 Baud
6	300 Baud

- Nur bei Sendung:

F2 - F7	Fix-Texte senden
F8	Datum und Uhrzeit (UTC) senden

Anschluß eines RTTY-Konverters an den USER-Port:

Pin		
B	CB1	Empfang
C	PB0	"
M	CB2	Sendung
N	GND	Masse

CB1 und PB0 sind parallelzuschalten (Drahtbrücke zwischen Pin B und Pin C), siehe dazu auch Seite 134 im Programmier-Handbuch.

Erläuterungen zum Baudot-Code finden Sie im Anhang A.2.

PROGRAMM-EINZELHEITEN

1.) Name: RTTY

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Funkfernschreib-Programm

4.) Anzahl der Bytes: 1934

5.) Benutzte Variablen:

G	Index für die Baud-Rate
G(X)	Baud-Rate als Zahl
A	Index für Umwandlung Baudot --> ASCII
I	Schleifenzähler
BZ	Bu/Zi-Modus
Z	Anzahl der gesendeten Zeichen
PP	Statusvariable 1 = Subroutine, 0 = Normal
A$	ASCII-Zeichen von Tastatur
B$	Baudot-Zeichen von Kanal
A$(X)	ASCII-Code für Umwandlung Baudot --> ASCII
B$(X)	Baudot-Code für Umwandlung ASCII > Baudot
F$(X)	Fix-Texte
F	Index für Fix-Texte

Benutzte Adressen:

#1	enthält den Index für die Baud-Rate (G)
#659	Kontrollregister RS232 (CHR$(96+G))
#660	Befehlsregister RS232 (CHR$(0))
#665,666	enhalten die Zeitkonstante für die RS232-Schnittstelle

6.) Listing: s. Bild 1.4.1

```
0 REM RTTY
10 POKE1,0:REM 45 BAUD
20 G=PEEK(1):IFG=0THENG=1
30 OPEN2,2,0,CHR$(96+G)+CHR$(0)
40 G=PEEK(1):IFG=0THENG=1
50 OPEN1,2,1,CHR$(96+G)+CHR$(0)
60 IFPEEK(1)=0THENPOKE666,94
70 DIMA$(63):DIMB$(91)
80 PRINT"□"
90 INPUT"TIME (HHMMSS)";TI$:PRINT
100 INPUT"DATUM";D$:PRINT
110 PRINT"FIX-TEXTE EINGEBEN":FORJ=2TO7
120 PRINT:PRINT:PRINT"TEXT F";J;"■ ?"
130 GETA$:IFA$=""ORA$>CHR$(90)THEN130
140 IFA$=CHR$(13)THEN160
150 PRINTA$;:F$(J)=F$(J)+A$:GOTO130
160 NEXT
170 PRINT"□"
180 G=PEEK(1)
190 G(0)=45:G(1)=50:G(2)=75:G(3)=110:G(4)=134.5:G(5)=150:G(6)=300
200 RESTORE:FORI=0TO63:READA$(I):NEXT
210 FORI=1TO91:READB:B$(I)=CHR$(B):NEXT
220 IFST=0AND(PEEK(37150)AND64)THEN220
230 PRINT:PRINT" EMPFANG    "G(G)"■BAUD■":PRINT
240 GET#2,B$
250 GETA$:IFA$="■"THENBZ=0:REM CRSR DWN
260 REM
270 IFA$="■"THEN360:REM F1
280 IFA$="■"THENBZ=32:REM CRSR RI
290 IFA$<"7"ANDA$>="0"THENPOKE1,VAL(A$):RUN20
300 IFB$=""THENGOTO240
310 A=ASC(B$):IFA$(A)="CR"THENPRINTCHR$(13):GOTO240
320 IFA=27THENBZ=32:GOTO240
330 IFA=31THENBZ=0:GOTO240
340 PRINTA$(A+BZ);:GOTO240
350 REM SENDUNG
360 PRINT:PRINT" SENDUNG    "G(G)"■BAUD■":PRINT
370 GETA$:IFA$=""THEN370
380 IFA$="■"THENPRINT:GOTO220:REM F1
390 IFA$="■"THENF=2:GOTO550:REM F2
400 IFA$="■"THENF=3:GOTO550:REM F3
410 IFA$="■"THENF=4:GOTO550:REM F4
420 IFA$="■"THENF=5:GOTO550:REM F5
430 IFA$="■"THENF=6:GOTO550:REN F6
440 IFA$="■"THENF=7:GOTO550:REM F7
450 IFA$="■"THENF=8:GOTO550:REM F8
460 IFASC(A$)>91THEN370
470 IFA$=CHR$(13)THENPRINT#2,CHR$(8);CHR$(2);:Z=0:PRINT:IFPP=0THEN370
480 IFA$="■"THENPRINT#2,CHR$(31);:BZ=32:GOTO370
490 IFA$="■"THENPRINT#2,CHR$(27);:BZ=0:GOTO370
500 PRINTA$;:Z=Z+1:IFZ=60THENA$=CHR$(13):GOTO470
510 IFA$<"A"ANDBZ=32THENPRINT#2,CHR$(27);:BZ=0
520 IFA$>"@"ANDBZ=0THENPRINT#2,CHR$(31);:BZ=32
530 PRINT#2,B$(ASC(A$));:IFPP=0THEN370
540 RETURN
550 F$(8)=D$+CHR$(13)+LEFT$(TI$,2)+":"+MID$(TI$,3,2)+" UTC"
560 IFLEN(F$(F))=0THEN370
570 PP=1
580 A$=CHR$(13):GOSUB470
590 FORJ=1TOLEN(F$(F))
600 A$=MID$(F$(F),J,1)
610 GOSUB470
620 NEXT
630 PP=0:GOTO370
```

Bild 1.4.1 (Teil 1)

```
640 DATA$,E,"",A," ",S,I,U,CR,D,R,J,N,F,C
650 DATAK,T,Z,L,W,H,Y,P,Q,O,B,G
660 DATA"",M,X,V,"","",3,"",-," ",',8,7,CR,$
670 DATA4,*,",",!,":",(,5,"'",),2,"",6,0,1,9,?,&,""
680 DATA.,/,=,""
690 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
700 DATA0,0,0,0,0,0,0,0,0,0,0,4,0,5,9,0,0,0,28,15
710 DATA18,0,17,12,3,28,29,22,23,19,1,10,16,21,7,6,24,14,11,0
720 DATA30,0,25,0,3,25,14,9,1,13,26,20,6,11,15,18,28,12,24,22
730 DATA23,10,5,16,7,30,19,29,21,17,8,2,31,27,4,0
```

Bild 1.4.1 (Teil 2)

7.) Erläuterungen zum Programm:

Zeile 10:
Der Index für die Baud-Rate (hier 45.45 Bd) wird in eine unbenutzte Zero-Page-Adresse geschrieben.

Zeile 20 - 60:
G übernimmt den Index für die Baud-Rate aus Adresse 1. Der Lesekanal (#2) und der Schreibkanal (#1) werden geöffnet. Das Bit-Muster für CHR$(96+G) sieht bei G=1 folgendermaßen aus:

Bit	7	6	5	4	3	2	1	0
	0	**1**	**1**	**0**	**0**	**0**	**0**	**1**

Die Beschreibung der einzelnen Bits finden Sie im Programmier-Handbuch auf Seite 131. In unserem Fall bedeutet es:
1 Datenwort ist 5 Bit breit plus 1 Stop-Bit, die Baud-Rate ist 50 Bd. Das Bit-Muster für CHR$(0) ist:

Bit	7	6	5	4	3	2	1	0
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**

und bedeutet: Vollduplex-Betrieb und keine Paritätsprüfung.

Um die auf den Amateurfunkbändern übliche Baud-Rate von 45.45 Bd zu realisieren, sind Kenntnisse über den Programmablauf beim öffnen der RS232-Schnittstelle notwendig. Hier sei dazu nur folgendes gesagt:
Nach OPEN steht in den Adressen #665 und #666 bzw. $0294 und $0295 die Zeitkonstante K für 1 Bit. Für jedes zu sendende oder zu empfangende Bit wird diese Konstante K dem Timer 1 des VIA #1 übergeben. Dieser zählt diese Konstante mit einer

Taktfrequenz von f=1.1 MHz auf 0 herunter.
Die Konstante K läßt sich folgendermaßen berechnen:

K = f/BR (BR = Baud-Rate)

Bei 50 Bd ist K = 22000 (DEZ) = 56BA (HEX).
Bei 45.45 Bd ist K = 24200 (DEZ) = 5E8A (HEX).
Die so errechnete Zeitkonstante muß für 45.45 Baud in die Adressen #665,666 gePOKEd werden (Zeile 60). In unserem Fall wird nur das HIGH-Byte (#666) geändert, denn die Differenz der LOW-Bytes ist vernachlässigbar klein.
Nach der oben beschriebenen Methode lassen sich alle Baud-Raten realisieren. Bei Baud-Raten großer als 300 Bd sollten aber Maschinen-Programme zum Auslesen der Pufferspeicher benutzt werden, um einen Datenverlust zu vermeiden (s. Programmier-Handbuch, Beschreibung der RS232-Schnittstelle).

Zeilen 70 - 210:
Die Variablen zur Umwandlung von ASCII nach Baudot und umgekehrt werden dimensioniert und mit der Umwandlungstabelle in den Zeilen 640-730 geladen.
Die Fix-Texte werden eingegeben und die restlichen Variablen werden initialisert.

Zeile 220:
Warten, bis kein Zeichen mehr im Ausgabepuffer steht

Zeilen 230 - 340:
Das erste Zeichen wird mit GET#2 aus dem Puffer des Kanals #2 gelesen und auf Gültigkeit geprüft. Gültige Zeichen werden in ASCII-Code umgewandelt und auf dem Bildschirm dargestellt.
Die Tastatur wird abgefragt und eventuell eingegebene Befehle werden ausgeführt.
Befehle können sein:
- Umschalten auf Sendung
- Umschalten in Bu-Modus
- Umschalten in Zi-Modus

Zeilen 350 - 630:
Die Tastatur wird abgefragt und das eingegebene Zeichen geprüft. Gültige Zeichen werden in den Baudot-Code umgewandelt und über den Kanal #1 ausgesendet. Gültige Befehle werden ausgeführt.

Befehle können sein:
- Umschalten auf Empfang
- Umschalten in Bu-Modus
- Umschalten in Zi-Modus
- Aussenden der Fix-Texte

Das Aussenden der Fix-Texte geschieht in der Subroutine ab Zeile 550.

Zeilen 640 - 730:
enthalten die Umwandlungstabelle ASCII --> Baudot und umgekehrt in DATA-Zeilen.
Auf die Umwandlung von ASCII --> Baudot und umgekehrt soll noch etwas näher eingegangen werden. Nehmen wir deshalb an:

A$ = "A" (Zeile 370)

Der ASCII-Code für "A" ist #65. In Zeile 460 wird geprüft, ob der ASCII-Code grüßer #91 ist, was ein im Baudot-Code nicht darstellbares Zeichen wäre. In Zeile 500 wird das A auf dem Bildschirm ausgegeben. In Zeile 530 wird B$(65) in den Kanal #2 geschrieben. B$(65) oder B$(ASC(A$))enthält den Baudot-Code für "A".
In gleicher Reihenfolge geschieht die Umwandlung Baudot-Code --> ASCII-Code ab Zeile 240.

1.5 FERNSCHREIBER ALS DRUCKER

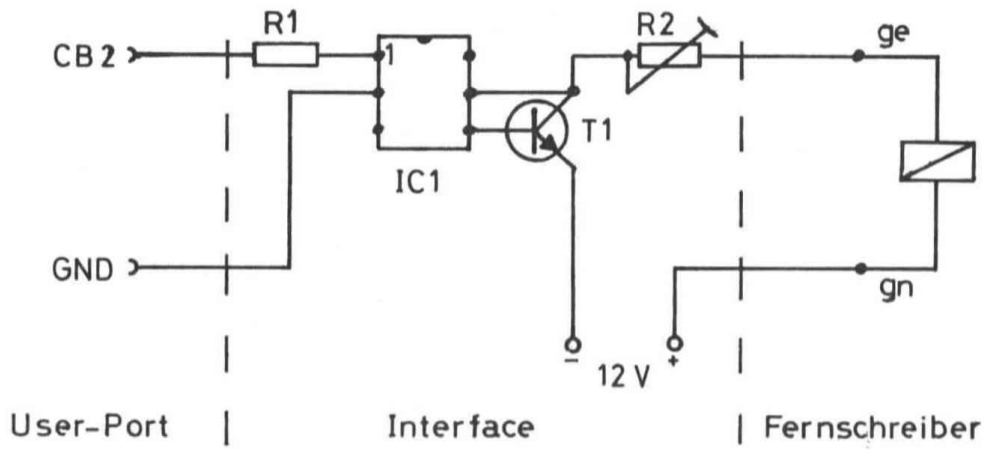
Die Idee, einen gebrauchten von Firmen oder Behörden ausgemusterten Fernschreiber anstelle eines teuren Druckers an den VC-20 anzuschließen, ist nicht neu. Wenn man von einigen Einschränkungen wie z.B. dem begrenzten Zeichenvorrat des Fernschreibers absieht, so sind diese Maschinen für unsere Zwecke gut geeignet. Software zum Betrieb eines Fernschreibers am VC-20 gibt es schon sehr viel, aber alle Programme haben einen entscheidenden Nachteil, sie sind nämlich sehr umständlich zu bedienen.

Das in diesem Kapitel vorgestellte PGM bietet Ihnen den gleichen Bedienungskomfort, wie ihn die Besitzer eines Druckers kennen. Zur Ausgabe des Baudot-Codes auf den Fernschreiber benutzt das PGM die im VC-20 implementierte V.24-(RS232-)Schnittstelle. Die

1.5 Fernschreiber als Drucker

OPEN-Anweisung ist in dem M-PGM eingebaut und zum besseren Verständnis des M-PGM-Teils 1 sei hier der Hinweis auf die im Programmier-Handbuch beschriebenen Betriebssystemroutinen erlaubt. Ab Seite 138 werden die verwendeten Routinen ausführlich beschrieben.

Die in Bild 1.5.1 gezeigte Schaltung für ein Interface zum Anschluß eines Fernschreibers an den VC-20 ist erprobt und schon mehrfach im Einsatz. Die Abgleicharbeiten beschränken sich auf die Einstellung des Linienstromes von 40 mA. Wenn Sie andere als die beschriebenen Bauteile verwenden, kann es passieren, daß Sie den Linienstrom von 40 mA nicht erreichen. Das kann zu Fehlschriften führen. Abhilfe schafft dann die Erhöhung der Spannung von 12 V auf 24 V.



R1 = 1 KΩ
R2 = 100 Ω (Trimmpoti)
IC1 = IL111 (Optokoppler)
T1 = BF459

Bild 1.5.1

BEDIENUNGSHINWEISE

Alle in dieser Bedienungsanleitung beschriebenen Anweisungen können im Direkt-Modus sowie in Basic-Programmen verwendet werden. Die OPEN-Anweisung muß allerdings immer in der 1. Zeile eines B-PGMs stehen.

SYS (Startadresse)	Diese Anweisung ist anstelle der OPEN-Anweisung zu verwenden. Ein File mit der Nummer #4 wird eröffnet. Die Startadresse wird nach Start des Basic-Loaders angezeigt.
PRINT #4,X	Gibt den Inhalt der Variablen X auf den Fernschreiber aus.
PRINT #4,A$	Gibt den Inhalt der String-Variablen A$ auf den Fernschreiber aus.
PRINT #4,"TEST"	Gibt eine Zeichenkette auf den Fernschreiber aus.

Komma und Semikolon haben bei den beschriebenen Anweisungen die gewohnten Tabulatorfunktionen.

CMD 4	Überträgt die Datenausgabe, die normalerweise auf dem Bildschirm erfolgen würde, auf den Fernschreiber (z.B. LIST). PRINT #4 überträgt die Datenausgabe wieder auf den Bildschirm. PRINT #4,"CLOSE4" überträgt die Datenausgabe wieder auf den Bildschirm und schließt gleichzeitig das File #4.
CLOSE 4	Schließt das File #4 und beendet die Datenausgabe auf den Fernschreiber.

Bevor Sie die CLOSE-Anweisung verwenden, sollten Sie sicherstellen, daß keine Zeichen mehr im Ausgabepuffer der V.24-Schnittstelle stehen. Im Direkt-Modus ist es einfach. Sie brauchen nur zu warten, bis der Fernschreiber nicht mehr druckt. In B-PGMen

können Sie folgende Zeile verwenden, um festzustellen, ob der Ausgabepuffer leer ist. Diese Zeile ist immer vor der CLOSE-Anweisung einzugeben.

```
100 IF ST=0 AND (PEEK(37150) AND 64) THEN 100
110 CLOSE 4
```

Die im Programmier-Handbuch auf Seite 134 abgedruckte Basic-Zeile zur Prüfung des Ausgabepuffers ist falsch.

PROGRAMM-EINZELHEITEN

1.) Name: TTY

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Betriebssystem-Zusatzroutine

4.) Anzahl der Bytes
des Basic-Loaders: 1157
des M-PGMs: 176

5.) Benutzte Adressen:

$00	Zwischenspeicher für Akku (ASCII-Code)
$01	Zwischenspeicher für X-Register
$02	Flag auf Bu/Zi-Modus
$03	Zwischenspeicher für Akku (Baudot-Code)

6.) Listings
des Basic-Loaders: siehe Bild 1.5.2
des M-PGMs: siehe Bild 1.5.3

7.) Erläuterungen zum M-PGM:

Teil 1: Zeilen 2002 - 2021
Dieser PGM-Teil eröffnet die V.24-Schnittstelle und weist dieser die Filenummer #4 zu. Die Baudrate wird auf 50 Bd eingestellt und die Breite des auszugebenden Zeichens wird auf 5 Bit festgelegt. Der Vektor "Ausgabe eines Zeichens" wird geändert und zeigt jetzt auf Teil 2 des M-PGMs.

```
0 POKE56,PEEK(56)-1
1 A=PEEK(56)*256:E=A+175:S=A+2
2 PRINT"■■■ M-PGM MUSS AM ANFANG
4 PRINT"■ EINER PAGE ABGELEGT
6 PRINT"■ WERDEN.
10 PRINT"■ FOLGENDE DATA-ZEILEN
12 PRINT"■ SIND JE NACH AUSBAU
14 PRINT"■ ABZUAENDERN:
15 PRINT
18 FORI= A TO E :READDC:IFDC<256THEN20
19 DC=PEEK(56):PRINT" ZEILE"PEEK(63)+PEEK(64)*256
20 POKEI,DC:NEXT
21 PRINT:PRINT"STARTADRESSE="S
22 DATA97,0,169,4
24 DATA162,2,160,1
26 DATA32,186,255
28 DATA169,2,162,0,160
30 DATA1000:REM*********  PAGE ???
32 DATA32,189,255,32,192
34 DATA255,169,34
36 DATA141,38,3,169
37 DATA1000:REM ********  PAGE ???
38 DATA141,39,3
40 DATA96,133,0,165
42 DATA154,201,2,240
44 DATA5,165,0,76
46 DATA122,242,165
48 DATA0,201,13,208
50 DATA15,169,8,32
52 DATA186,242,169
54 DATA2,32,186,242
56 DATA165,0,96,234
58 DATA234,134,1,165
60 DATA0,41,63,170
62 DATA189,112
63 DATA1000:REM ********   PAGE ???
64 DATA133,3,41,32
66 DATA197,2,240,14
68 DATA133,2,170,240
70 DATA4,169,27,208
72 DATA2,169,31,32
74 DATA186,242,165
76 DATA3,32,186,242
78 DATA165,0,166,1
80 DATA96,234,234
82 DATA0,3,25,14,9
84 DATA1,13,26,20
86 DATA6,11,15,18
88 DATA28,12,24,22
90 DATA23,10,5,16
92 DATA7,30,19,29
94 DATA21,17,0,0,0
96 DATA0,0,4,0,37
98 DATA45,0,0,0,37
100 DATA47,50,60,49
102 DATA44,35,60,61
104 DATA54,55,51,33
106 DATA42,48,53,39
108 DATA38,56,46,0
110 DATA0,62,0,57
```

Bild 1.5.2

Teil 2: Zeilen 2022 - 202C
In diesem PGM-Teil wird geprüft, ob die Ausgabe des Zeichens über die V.24-Schnittstelle erfolgen soll (die Adresse $9A enthält die momentane Gerätenummer). Wenn ja, dann erfolgt der Sprung in Teil 3 des PGMs, andernfalls erfolgt der Sprung in die Betriebssystemroutine "Ausgabe eines Zeichens" (JMP $F27A).

Teil 3: Zeilen 202F - 206F
Dieser Teil des PGMs ist die eigentliche Routine zur Ausgabe eines Zeichens auf den Fernschreiber. Hier wird der ASCII-Code in den Baudot-Code umgewandelt und das auszugebende Zeichen in den Ausgabepuffer der V.24-Schnittstelle geschrieben. An Adresse $2031 wird geprüft, ob es sich bei dem auszugebenden Zeichen um ein CR (Carrige-Return = Wagenrücklauf) handelt. Wenn ja, dann wird der Baudot-Code für WR (Wagenrücklauf) und ZL (Zeilenvorschub) in den Ausgabepuffer geschrieben (JSR $F2BA).
Die Routine ab Adresse $F2BA ist eine Betriebssystem-Routine des VC-20 und wird im Anhang A.4 eingehend beschrieben.
Nach Rückkehr aus dieser Routine erfolgt der Rücksprung in das Basic-PGM bzw. in die "Eingabe-Warteschleife". Die beiden NOP's hinter RTS sind Platzhalter, denn anstelle des RTS kann auch der Befehl JMP $E742 eingefügt werden. JMP $E742 ist ein Sprung in die Betriebssystem-Routine "Ausgeben eines Zeichens auf den Bildschirm" und bewirkt, daß die Zeichen, die auf dem Fernschreiber abgedruckt werden, auch auf dem Bildschirm dargestellt werden.
Wenn es sich bei dem auszugebenden Zeichen nicht um ein CR handelt, dann wird ab Adresse $2044 mit der Umwandlung in den Baudot-Code angefangen. Das X-Register wird nach Adresse $01 gerettet. Der Akku wird mit $3F logisch UND-verknüpft und damit das auszugebende Zeichen auf Buchstaben, Ziffern und Satzzeichen begrenzt (vgl. Tabelle der Bildschirm-Codes im VC-20-Handbuch, Seite 141). An Adresse $204B wird der Baudot-Code für das auszugebende Zeichen aus der Tabelle ab Adresse $2070 gelesen und an Adresse $03 zwischengespeichert. Da der Baudot-Code nur 5 Bit breit ist, können die verbleibenden 3 Bit anderweitig benutzt werden. In unserem Fall ist das 6. Bit ein Flag für den Bu/Zi-Modus. Wenn das 6. Bit 0 ist, dann handelt es sich um einen Buchstaben; wenn es 1 ist, dann handelt es sich um ein Satzzeichen.

```
2000 61 00
2002 LDA ##$04
2004 LDX ##$02
2006 LDY ##$01
2008 JSR $FFBA
200B LDA ##$02
200D LDX ##$00
200F LDY ##$5F
2011 JSR $FFBD
2014 JSR $FFC0
2017 LDA ##$22
2019 STA $0326
201C LDA ##$5F
201E STA $0327
2021 RTS
```

Teil 1

```
2022 STA $00
2024 LDA $9A
2026 CMP ##$02
2028 BEQ $202F
202A LDA $00
202C JMP $F27A
```

Teil 2

```
202F LDA $00
2031 CMP ##$0D
2033 BNE $2044
2035 LDA ##$08
2037 JSR $F2BA
203A LDA ##$02
203C JSR $F2BA
203F LDA $00
2041 RTS
2042 NOP
2043 NOP
2044 STX $01
2046 LDA $00
2048 AND ##$3F
204A TAX
```

Teil 3

```
204B LDA $5F70,X
204E STA $03
2050 AND ##$20
2052 CMP $02
2054 BEQ $2064
2056 STA $02
2058 TAX
2059 BEQ $205F
205B LDA ##$1B
205D BNE $2061
205F LDA ##$1F
2061 JSR $F2BA
2064 LDA $03
2066 JSR $F2BA
2069 LDA $00
206B LDX $01
206D RTS
206E NOP
206F NOP
```

Forts. Teil 3

```
2070 00 03 19 0E 09
2075 01 0D 1A 14 06
207A 0B 0F 12 10 0C
207F 18 16 17 0A 05
2084 10 07 1E 13 1D
2089 15 11 00 00 00
208E 00 00 04 00 25
2093 2D 00 00 00 25
2098 2F 32 3C 31 2C
209D 23 3C 3D 36 37
20A2 33 21 2A 30 35
20A7 27 26 38 2E 00
20AC 00 3E 00 39
```

Teil 4

Bild 1.5.3

An Adresse $2050 wird der Inhalt des Akku mit $20 logisch UND-verknüpft und mit dem Inhalt der Adresse $02, die ebenfalls das Flag für den Bu/Zi-Modus enthält, verglichen. Wenn die Flags ungleich sind, so wird das neue Flag in Adresse $02 geschrieben und der Baudot-Code für Bu, bzw. Zi in den Ausgabepuffer geschrieben (JSR $F2BA).
An Adresse $2064 wird der zwischengespeicherte Baudot-Code in den Akku geladen und in den Ausgabepuffer geschrieben (JSR $F2BA). Ab Adresse $2069 werden Akku und X-Register mit den geretteten Daten geladen und es erfolgt der Rücksprung in das

B-PGM, bzw. in die Eingabe-Warteschleife. Die beiden NOP's sind wiederum Platzhalter für den Befehl JMP $E742. falls Sie die Zeichen auch auf dem Bildschirm darstellen wollen.

Teil 4: Zeilen 2070 - 20AC
Dieser PGM-Teil enthält die Tabelle zur Umwandlung des ASCII-Codes in den Baudot-Code. Für Zeichen im ASCII-Code, die im Baudot-Code nicht darstellbar sind, enthält die Tabelle jeweils eine 0.

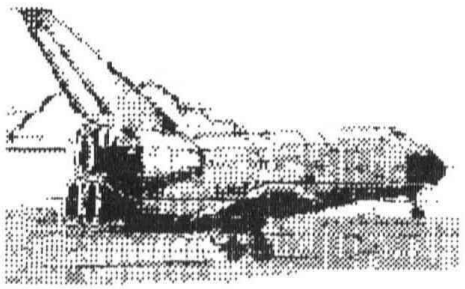
1.6 BILDSCHIRMINHALT DRUCKEN: SCREEN COPY

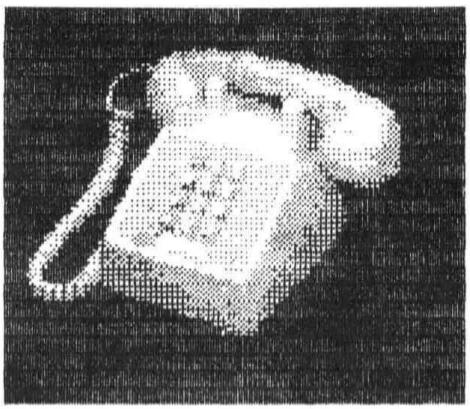
SCREEN COPY ist ein Hilfs-PGM, das es erlaubt, selbsterstellte Bildschirmgrafiken auf einen angeschlossenen Drucker (hier VC-1525 bzw. GP100-VC) zu kopieren. Bedingt durch die vom Drucker darstellbare Zeichengroße von nur 7x6 Punkten, ist bei der PGM-Erstellung etwas "geschludert" worden: die 8. Reihe des 8x8 Punkte großen Zeichens auf dem Bildschirm wird einfach vernachlässigt und nicht auf dem Drucker dargestellt. Texte wirken dadurch etwas gedrückt, den ausgedruckten Graphiken aber sieht man die fehlende 8. Reihe nicht an (siehe Beispiele in Bild 1.6.1).

BEDIENUNGSHINWEISE

Das PGM muß an vorhandene PGMe angehangt werden. Nachdem die Grafik erstellt ist (z.B. mit dem Graphik-Modul VC 1211), wird es mit GOTO 60000 gestartet. Das PGM errechnet sich dann alle benötigten Werte, wie z.B. die Bildschirmgröße und die Lage des Bildschirmspeichers selbst, lediglich für die Kopie einer Bildschirmgraphik, die aus 8x16 Punkten je Zeichen besteht, müssen die folgenden vier Zeilen geandert bzw. in das PGM eingefugt werden.

```
60080 A=PEEK(I)*2
60100 SZ=SZ+1:IF SZ=S THEN PRINT#4,A$:A$="":
      PRINT#4,B$:B$'"":SZ=0
60171 IF PEEK(K+8) AND X THEN ZZ=ZZ+Y
60201 B$=B$+CHR$(ZZ OR 128):ZZ=0
```





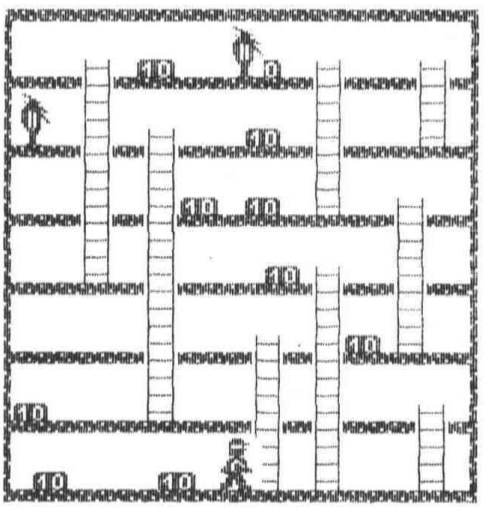


Bild 1.6.1

PROGRAMM-EINZELHEITEN

1.) Name: SCREEN COPY

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Hilfsprogramm

4.) Anzahl der Bytes: 409

5.) Benutzte Variablen:

S	Anzahl der senkrechten Spalten
ZG	Startadresse des Zeichengenerators (ROM/RAM)
BA	Anfangsadresse des Video-Speichers
BE	Endadresse des Video-Speichers
I,J,K	Laufvariablen
A	Bildschirm-Code des umzuwandelnden Zeichens

```
60000 REM SCREEN COPY EINFACHE ZEICHENHOEHE
60010 OPEN4,4:PRINT#4,CHR$(8)
60020 S=PEEK(36866)AND127
60030 IF NOT PEEK(36869) AND 8 THEN ZG=32768
60040 ZG=ZG+(PEEK(36869)AND 7)*1024
60050 BA=PEEK(648)*256
60060 BE=BA+(PEEK(36867)AND126)/2*S
60070 FORI=BA TO BE
60080 A=PEEK(I)
60090 GOSUB60130
60100 SZ=SZ+1:IFSZ=STHENPRINT#4,A$:A$="":SZ=0
60110 NEXT
60120 CLOSE4:END
60130 X=128
60140 FORJ=1TO8
60150 Y=1
60160 FORK=ZG+A*8TOZG+A*8+7
60170 IFPEEK(K)AND X THEN Z=Z+Y
60180 Y=Y*2
60190 NEXT
60200 A$=A$+CHR$(Z OR 128):Z=0
60210 X=X/2
60220 NEXT
60230 RETURN
```

Bild 1.6.2

SZ	Spaltenzähler
A$	Bildschirmzeile im Drucker-Graphik-Code
X	Zeiger auf eine Spalte
Y	Wert eines Bits
Z	Drucker-Graphik-Code für eine Spalte des umzuwandelnden Zeichens

6.) Listings der PGMe zum Ausdruck

von 8x8-Bit-Zeichen: Bild 1.6.2

von 8x16-Bit-Zeichen: Bild 1.6.3

```
60000 REM SCREEN COPY DOPPELTE ZEICHENHOEHE
60010 OPEN4,4:PRINT#4,CHR$(8)
60020 S=PEEK(36866)AND127
60030 IF NOT PEEK(36869) AND 8 THEN ZG=32768
60040 ZG=ZG+(PEEK(36869)AND 7)*1024
60050 BA=PEEK(648)*256
60060 BE=BA+(PEEK(36867)AND126)/2*S
60070 FORI=BA TO BE
60080 A=PEEK(I)*2
60090 GOSUB60130
60100 SZ=SZ+1:IFSZ=STHENPRINT#4,A$:A$="":PRINT#4,B$:B$="":SZ=0
60110 NEXT
60120 CLOSE4:END
60130 X=128
60140 FORJ=1TO8
60150 Y=1
60160 FORK=ZG+A*8TOZG+A*8+7
60170 IFPEEK(K)AND X THEN Z=Z+Y
60171 IFPEEK(K+8)AND X THEN ZZ=ZZ+Y
60180 Y=Y*2
60190 NEXT
60200 A$=A$+CHR$(Z OR 128):Z=0
60201 B$=B$+CHR$(ZZ OR 128):ZZ=0
60210 X=X/2
60220 NEXT
60230 RETURN
```

Bild 1.6.3

7.) Erläuterungen zum Programm:

Zeile 60010:
Drucker-Kanal öffnen und Graphik-Modus einschalten

Zeile 60020:
Variable S enthält die Anzahl der Bildschirmspalten (Kontrollregister 2 des Video-Chips (6561)

Zeile 60030:
Bit 3 des Kontrollregister 5 des Video-Chips gibt an, ob die Anfangsadresse des Zeichengenerator-Speichers im ROM- oder RAM-Bereich liegt.

ZG enthält 32768 wenn ROM
ZG enthält 0 wenn RAM

Zeile 60040:
Die Bits 0-2 des Kontrollregisters 5 enthalten den Zeiger auf die Anfangsadresse des Zeichengenerator-ROM/RAM-Bereichs.

Zeile 60050:
Die Adresse #648 enthält das H-Byte der Anfangsadresse des Video-Speichers. Die Anfangsadresse wird in die Variable BA geschrieben.

Zeile 60060:
Die Endadresse BE des Video-Speichers errechnet sich aus der Anfangsadresse BA, der Anzahl der Spalten S und der Anzahl der Reihen, die aus dem Inhalt der Bits 1-6 des Kontrollregisters 3 des Video-Chips errechnet werden.

Zeilen 60070 - 60090:
Der Inhalt des Video-Speichers wird Byte für Byte der Variablen A zugewiesen und in dem Unterprogramm ab Zeile 60130 in einen Graphik-Code zur Ausgabe auf den Drucker umgewandelt.

Zeile 60100:
Nach Rückkehr aus dem Unterprogramm wird die Variable SZ (Spaltenzähler) um 1 erhöht und mit der Anzahl der vorhandenen Spalten (S) verglichen. Wenn die Bedingung erfüllt ist (SZ=S), dann wird eine koplette Bildschirmzeile im Graphik-Code auf den Drucker ausgegeben und der Spaltenzähler zurückgestellt.

Zeile 60110: Nächstes Zeichen

Zeile 60120: Kanal schließen und PGM beenden

Zeile 60130:
Variable X enthält den Zeiger auf die zu prüfenden Spalten eines Zeichens.

Zeile 60140:
Die Schleife wird 8mal durchlaufen, um alle 8 Spalten eines Zeichens auszuwerten.

Zeile 60150:
Variable Y enthält den Wert des zu prüfenden Bits

Zeilen 60160 - 60180:
Die 8x8-Bit-Matrix eines Zeichens wird Bit für Bit geprüft und in den Drucker-Graphik-Code umgewandelt. Variable Z enthält die Summe der Wertigkeiten von gesetzten Bits in einer Spalte der 8x8-Bit-Matrix.

Zeile 60190: Nächstes Bit

Zeile 60200:
Der Drucker-Graphik-Code wird in die Variable A$ geschrieben. Dieser Graphik-Code muß mit 128 (Bit 7) logisch ODER-verknüpft werden, damit das Zeichen nicht als Befehl (CHR$(13) = CR) erkannt und ausgewertet wird.

Zeile 60210: Zeiger auf die nächste zu prüfende Spalte setzen

Zeile 60220: Nächste Spalte

Zeile 60230: Rücksprung aus dem Unterprogramm

1.7 ÄNDERUNG DER FLOPPY-NUMMER

Beim VC-20 sind bis zu 8 Floppy's 1540 bzw. 1541 anschließbar. Sollen alle diese per Programm oder auch im Direkt-Modus angesprochen werden können, so ist die gewohnte Benutzung der Gerätenummer 8 nicht mehr ausreichend.

Das Programm (s. Listing in Bild 1.7.1) verschafft Ihnen die Möglichkeit, Ihren angeschlossenen Floppy's unterschiedliche Gerätenummern (Device Numbers) zwischen 8 und 15 zuzuweisen. Auf de-

taillierte Erläuterungen zum Programmablauf werde hier verzichtet, weil es sich von selbst erklärt.

```
10 PRINT"┌──────────────────────┐";
11 PRINT"│                      │";
12 PRINT"│  CHANGE 1540/1541    │";
13 PRINT"│                      │";
14 PRINT"│  DOS 2.6  29/7/82    │";
15 PRINT"│                      │";
16 PRINT"│ DEVICE NUMMERN VON   │";
17 PRINT"│                      │";
18 PRINT"│      8 BIS 15        │";
19 PRINT"│                      │";
20 PRINT"│  NACH AUSSCHALTEN    │";
21 PRINT"│                      │";
22 PRINT"│ DER FLOPPY- DEVICE   │";
23 PRINT"│                      │";
24 PRINT"│  NUMMER WIEDER 8     │";
25 PRINT"└──────────────────────┘";
100 INPUT"ALTE DEVICE-NUMMER";DO
110 IFDO<8 OR DO>15 THEN100
140 PRINT
150 INPUT"NEUE DEVICE-NUMMER";DN
160 IFDN<8 OR DN>15 THEN150
200 OPEN15,DO,15:REM KOMMANDO KANAL
210 PRINT#15,"M-W"CHR$(119)CHR$(00)CHR$(2)CHR$(DN+32)CHR$(DN+64)
220 CLOSE15
230 END
```

Bild 1.7.1

Jedoch sollte noch erwähnt werden, wie bei der Geräteummernfestlegung selbst die einzelnen Floppy's unterschieden werden können. Dazu nutzen wir die Tatsache aus, daß rein hardwaremäßig nach Einschalten der Floppy dieser die Geratenummer 8 zugewiesen ist. Zuerst wird nur eine der angeschlossenen Floppy's eingeschaltet. Danach wird ihr mittels dem hier vorgestellten Programm eine Nummer ungleich 8 zugeteilt. Schaltet man nun die nächste Floppy ein, so sind beide Floppy's eindeutig adreßierbar, weil ja die neu hinzugekommene die Standard-Geratenummer 8 aufweist. Diese muß gleichfalls mittels Programm umgeändert werden, will man eine weitere Floppy zuschalten, etc., etc.

1.8 OLD FÜR DISKETTE = REGENERIEREN

Wie der Basic-Befehl NEW, so löscht auch der Disketten-Befehl SCRATCH nicht das PGM, sondern zerstört nur den Vektor auf das PGM. Diese "Zerstörung" geschieht durch das Löschen des File-Kennzeichens in dem Directory, wodurch die in der BAM als belegt gekennzeichneteten Blöcke nicht mehr geschützt sind und freigegeben werden. Der File-Name selbst und der Inhalt der Blocke werden nicht zerstört.

Das in diesem Kapitel beschriebene PGM macht nichts anderes, als die Blöcke, in denen das PGM steht, wieder als belegt zu kennzeichnen und mit dem File-Namen zu verknüpfen.

Im Anschluß an die Programm-Einzelheiten befindet sich daruberhinaus die auf die VC-20-Floppy bezogenen Fehlernummern.

BEDIENUNGSHINWEISE

Wenn Sie merken, daß unbeabsichtigt ein PGM auf der Diskette gelöscht wurde, z.B. durch SCRATCH zusammen mit dem Joker (*), dann laden Sie einfach das PGM "Regenerieren" und starten es. Das PGM fordert Sie auf, den Namen des gelöschten PGMs einzugeben. Dieser darf max. 16 Zeichen lang sein und den Joker dürfen Sie hierbei nicht verwenden. Anschließend wird das Directory nach dem Namen durchsucht und eine Fehlermeldung ausgegeben, wenn der Name nicht gefunden wurde, oder das PGM nicht mehr zu regenerieren ist, weil Sie zwischenzeitlich schon ein anderes PGM auf dieser Diskette geSAVEd haben. Nach erfolgreicher Regenerierung steht der PGM-Name in dem Directory und Sie können das PGM wieder laden.

PROGRAMM-EINZELHEITEN

1.) Name: REGENERIEREN

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Betriebssystem-Zusatzprogramm

4.) Anzahl der Bytes: 1084

5.) Benutzte Variablen:

NB$	Name des gelöschten PGMs
ND$	PGM-Namen, die im Directory stehen
B$	Übergabe-Variable: Block lesen
T	Spur (Track) der Directory
S	Sektor der Directory
B(0...255)	enhält alle Bytes eines Blocks
I	Laufvariable
TD	Spur des nächsten Blocks, in dem das PGM gespeichert ist
SD	Sektor des nächsten Blocks, in dem das PGM gespeichert ist
EN	Fehler-Nummer
EN$	Fehler-Name
ET	Fehler-Spur
ES	Fehler-Sektor

6.) Listing: siehe Bild 1.8.1

7.) Erläuterungen zum Programm:

Zeilen 10 - 60:
Der Name des gelöschten PGMs wird in die Variable NB$ geschrieben und auf seine Größe geprüft (der Eintrag des File-Namen in das Directory ist max. 16 Zeichen).

Zeile 70:
Die Variablen T (TRACK) und S (SECTOR) bekommen die Adresse des 1. Blocks des Directory zugewiesen. Variable B wird dimensioniert.

Zeile 80:
Der Befehls- und Fehlerkanal wird eröffnet.

Zeile 90:
Der Fehlerkanal wird ausgelesen. Bei einer Fehlermeldung wird das PGM mit dem Ausdruck der Fehlermeldung beendet.

Zeile 100:
Die Datenkanäle (#2 und #3) werden eröffnet.

Zeile 110:
Der File-Name (NB$) wird auf 16 Zeichen ergänzt.

Zeile 120:
Über den Befehlskanal wird der "B-R:"-Befehl zum Lesen des 1. Blocks des Directory gegeben.

```
0 REM REGENERIEREN
10 PRINT"■"
20 PRINT"■■ NAME DES GELOESCHTEN"
30 PRINT" PROGRAMMS"
40 PRINT
50 INPUTNB$
60 IFLEN(NB$)>16THENPRINT" NAME IST ZU LANG":GOTO20
70 T=18:S=1:DIMB(255)
80 OPEN15,8,15,"I0"
90 INPUT#15,EN,EN$,ET,ES:IFEN>0THENGOTO410
100 OPEN2,8,2,"#":OPEN3,8,3,"#"
110 NB$=NB$+CHR$(160):IFLEN(NB$)<16THEN110
120 PRINT#15,"U1:"2;0;T;S
130 FORI=0TO255:PRINT#15,"B-P:"2,I:GET#2,B$:IFB$=""THENB$=CHR$(0)
140 B(I)=ASC(B$):NEXT
150 FORI=2TO226STEP32:ND$="":FORJ=3TO18:ND$=ND$+CHR$(B(I+J)):NEXT
160 IFB(I)=130ANDND$=NB$THENGOTO400
170 IFND$=NB$ANDB(I)=0ORB(I)=128THEN210
180 NEXT
190 IFB(0)>0THENT=B(0):S=B(1):GOTO120
200 GOTO380
210 PRINT#15,"U1:"3;0;T;S
220 PRINT#15,"B-P:"3;I+1:GET#3,B$:IFB$=""THENB$=CHR$(0)
230 TD=ASC(B$)
240 PRINT#15,"B-P:"3;I+2:GET#3,B$:IFB$=""THENB$=CHR$(0)
250 SD=ASC(B$)
260 PRINT#15,"B-A:"0;TD;SD
270 INPUT#15,EN,EN$,ES,ET:IFEN>0THENGOTO360
280 PRINT#15,"U1:"3;0;TD;SD
290 PRINT#15,"B-P:"3;0:GET#3,B$:IFB$=""THENB$=CHR$(0)
300 TD=ASC(B$)
310 PRINT#15,"B-P:"3;1:GET#3,B$:IFB$=""THENB$=CHR$(0)
320 SD=ASC(B$)
330 IFTD>0THEN260
340 PRINT#15,"B-P:"2;I:PRINT#2,CHR$(130);
350 PRINT#15,"U2:"2;0;T;S:GOTO390
360 PRINT#15,"V0"
370 PRINT"IST NICHT MEHR ZU REGENERIEREN":GOTO420
380 PRINT"NICHT GEFUNDEN":GOTO420
390 PRINT"IST REGENERIERT":GOTO420
400 PRINT"WURDE NICHT GELOESCHT":GOTO420
410 PRINTCHR$(17)"DISK-ERROR :":PRINTEN,EN$,ET,ES
420 CLOSE15
```

Bild 1.8.1

Zeilen 130 - 140:
Der 1. Block des Directory wird über Kanal #2 gelesen und die einzelnen Bytes den Variablen B(0) bis B(255) zugewiesen. B(0) und B(1) enthalten den Vektor auf den nächsten Block des Directory.

Zeile 150:
Die Variablen B(0...255) werden nach dem File-Namen durchsucht.

Zeile 160:
Wenn der File-Name gefunden wird und als "nicht gelöscht" gekennzeichnet ist, dann wird das PGM beendet.

Zeile 170:
Wenn der File-Name gefunden wird und als "gelöscht" gekennzeichnet ist, dann wird mit der Regenerierung ab Zeile 210 begonnen.

Zeile 180:
Den nächsten File-Namen vergleichen

Zeile 190:
Wenn der File-Name im 1. Block des Directory nicht gefunden wurde, dann wird eine gültige Adresse des nächsten Blocks in die Variablen T und S übernommen (eine Adresse ist gültig, wenn die Nummer der Spur größer 0 ist). Ab Zeile 120 wird der nächste Block nach dem File-Namen durchsucht.

Zeile 200:
Bei einer ungültigen Blockadresse wird das PGM beendet.

Zeile 210:
Die Variable I enthält den Vektor auf die 1. Blockadresse des gelöschten PGMs. Diese Adresse soll über Kanal #3 ausgelesen werden.

Zeilen 220 - 250:
Die Adresse des 1. PGM-Blocks wird gelesen und den Variablen TD und SD zugewiesen.

Zeile 260:
Der gefundene Block wird in der BAM wieder als belegt gekennzeichnet, aber noch nicht mit einem File-Namen verknüpft.

Zeile 270:
Der Fehlerkanal wird ausgelesen. Wenn ein Fehler vorliegt, wird das PGM beendet (Zeile 360). Eine Fehlermeldung wird z.B. dann ausgegeben, wenn der Block schon als belegt gekennzeichnet war. Das kann dann der Fall sein, wenn nach dem Löschen des PGMs ein anderes PGM geSAVEd wurde.

Zeilen 280 - 330:
Der Vektor auf den nachsten Block des PGMs wird gelesen und ab Zeile 260 in die BAM geschrieben. Eine ungültige Blockadresse ist hierbei der Zeiger auf das Ende des PGMs.

Zeilen 340 - 350:
Die in der BAM als belegt gekennzeichneten Blöcke werden jetzt mit einem File-Namen verknüpft und das PGM beendet.

Zeile 360:
Der VALIDATE-Befehl wird ausgeführt.

Zeilen 370 - 400:
Bei PGM-Ende wird eine Aussage uber den Verlauf der Regenerierung gemacht.

Zeile 410:
Druckt den Inhalt des Fehlerkanals

Zeile 420:
Schließt alle Kanäle

LISTE DER FEHLERNUMMERN UND BESCHREIBUNGEN

0 : Kein Fehler

1 : Rückmeldung für Filelöschung (keine Fehlermeldung)

20 : READ ERROR
Header des gesuchten Datenblocks wird nicht gefunden

21 : READ ERROR
SYNC-Markierung wird nicht gefunden. Gründe dafür können schlechte Justierung des Schreib/Lesekopfes, nicht formatierte Disketten oder schlechter Sitz der Diskette sein. Kann auch Hardware-Defekt anzeigen.

22 : READ ERROR
Dieser Fehler tritt im Zusammenhang mit den BLOCK-Befehlen auf und zeigt an, daß ein Block gelesen oder geprüft (verified) werden sollte, der nicht ordnungsgemäß geschrieben wurde.

23 : READ ERROR
Die Daten, die in den DOS-Speicher geschrieben wurden, enthalten einen Prüfsummenfehler, d.h. ein Datenbyte oder mehrere sind fehlerhaft. Dieser Fehler kann auch Erdungsprobleme anzeigen.

24 : READ ERROR
Die Daten oder der Header wurden in den DOS-Speicher eingelesen, aber i;n den Datenbytes existieren fehlerhafte Bitmuster. Dieser Fehler kann auch Erdungsprobleme anzeigen.

25 : WRITE ERROR
Fehlende Übereinstimmung zwischen den Daten im DOS-Memory und den Daten auf der Diskette.

26 : WRITE PROTECT ON
Es wird versucht, auf eine Diskette zu schreiben, die mit einem Schreibschutz versehen ist.

27 : READ ERROR
Es liegt ein Fehler im Header des zu lesenden Datenblocks vor (Prüfsumme stimmt nicht). Dieser Fehler kann auch Erdungsprobleme anzeigen.

28 : WRITE ERROR
Nach dem Schreiben eines Datenblocks sucht der Controller die SYNC-Zeichenfolge des nächsten Datenblocks. Wenn er diese Synchronisationsmarkierung nicht innerhalb eines bestimmten Zeitraums findet, wird diese Fehlermeldung ausge-

geben. Grund dafür kann z.B. ein zu langer Datenblock oder ein Hardwarefehler sein.

29 : DISK ID MISMATCH
Die im DOS-Speicher vorhandene ID stimmt nicht mit der ID auf der Diskette überein. Die Diskette wurde nicht initialisiert, oder der Header ist fehlerhaft.

30 : SYNTAX ERROR
Das DOS kann den über den Befehlskanal geschickten Befehl nicht interpretieren.

31 : SYNTAX ERROR
Befehl wird nicht erkannt

32 : SYNTAX ERROR
Befehl ist länger als 58 Zeichen

33 : SYNTAX ERROR
Falsche Verwendung des "Jokers" im OPEN- oder SAVE-Befehl

34 : SYNTAX ERROR
Filename fehlt oder kann vom DOS nicht erkannt werden. Typischer Fehler: ein Doppelpunkt (:) fehlt.

39 : SYNTAX ERROR
Befehl wird nicht erkannt

50 : RECORD NOT PRESENT
Die Fehlermeldung wird ausgegeben, wenn versucht wird, nach den letzten aufgezeichneten Daten mit INPUT# oder GET# weiterzulesen.

51 : OVERFLOW IN RECORD
Die Anzahl der Zeichen (inkl. CR) ist größer als die vorgegebene Recordlänge

52 : FILE TOO LARGE
Recordnummer (in einem relativen File) ist zu groß; das Fassungsvermögen der Diskette wird überschritten.

60 : WRITE FILE OPEN
Ein Schreibfile, das nicht geschlossen wurde, soll geöffnet werden.

61 : FILE NOT OPEN
Es soll ein File angesprochen werden, das nicht geöffnet wurde.

62 : FILE NOT FOUND
File existiert nicht.

63 : FILE EXISTS
Ein File soll einen Namen erhalten, der bereits auf der Diskette existiert.

64 : FILE TYPE MISMATCH
File-Typ stimmt nicht mit dem Directory-Eintrag überein.

65 : NO BLOCK
Diese Fehlermeldung steht im Zusammenhang mit dem B-A-Befehl. Der Block, der als belegt gekennzeichnet werden soll, ist bereits belegt. Die Parameter, die ausgegeben werden, geben Spur und Sektor des nächsten frei verfügbaren Blocks an. Werden zwei Nullen ausgegeben, so sind alle Blocks mit höheren Nummern belegt.

66 : ILLEGAL TRACK AND SECTOR
Das DOS findet nicht den Vektor zum nächsten Block.

67 : ILLEGAL T OR S
Ungültige Spur- oder Sektornummer.

70 : NO CHANNEL AVAILABLE
Ein direkt angespochener Kanal ist bereits belegt oder (falls kein bestimmter Kanal angesprochen wurde) kein Kanal mehr ist frei.

71 : DIR ERROR
Die BAM kann nicht gelesen werden, was eventuell seinen Grund darin haben kann, daß sie im DOS-Speicher überschrieben wurde. Ein Reinitialisieren der Diskette kann diesen Fehler in manchen Fällen beheben.

72 : DISK FULL
Alle 664 Blocks der Diskette sind belegt, oder die Directory hat die maximale Anzahl von 144 Eintragen.

73 : CBM DOS V2.6 V170
Diese Meldung erscheint direkt nach dem Einschalten bzw. dann, wenn sie mit der VC1540 auf eine Diskette schreiben wollen, die mit einer anderen DOS-Version formatiert wurde. Auf COMMODORE-Rechnern existieren vier verschiedene DOS-Versionen:

DOS 1.0 : CBM 2040/3040
DOS 2.0 : CBM 4040
DOS 2.5 : CBM 8050
DOS 2.6 : CBM 1540

Voll kompatibel sind die Versionen 2.0 und 2.6. Mit DOS 1.0 und 2.0 bzw. 1.0 und 2.6 formatierte Disketten konnen wechselseitig gelesen, aber nicht beschrieben werden. Bei einem Schreibversuch erscheint die obige Fehlermeldung. Die DOS-Version 2.5 ist mit allen anderen Versionen weder schreib- noch lesekompatibel.

74 : DRIVE NOT READY
Es wurde versucht, die Floppy anzusprechen, ohne eine Diskette in das Laufwerk einzulegen.

1.9 CURSOR-STEUERUNG PER JOYSTICK

Schließt man einen Joystick an den VC-20 an, so bleibt es (leider) dem Benutzer bzw. Programmierer überlassen, die jeweilige Richtung, in die der Stab bewegt wird, zu bestimmen und im Programm zu verwerten. Das VC-20-Betriebssystem stellt keine Befehle zur Verfügung, die Richtung und das Drücken des Feuerknopfes festzustellen.

Es gibt wahrscheinlich kein VC-20-Buch, das sich diesem Problem nicht in irgendeiner Form gewidmet hat. Die hier vorgestellten Programme JOYSTICK-CURSOR 1 und 2 (Bilder 1.9.1 und 1.9.2) allerdings weisen 2 Besonderheiten auf:

```
0 REM LIES CURSOR PER JOYSTICK
10 GOSUB5000
20 PRINTCU-UG:GOSUB5030:GOTO20
5000 BP=PEEK(648):CL=37888+(BPAND2)*256
5010 POKE37139,PEEK(37139)AND(255-4-8-16-32)
5020 DF=506:CU=BP*256:UG=CU:OG=UG+505
5030 GOSUB10000:RI=ABS((RI-100)/4)-7
5040 IFPEEK(CU)>127THENPOKECU,PEEK(CU)-128
5050 IFF=0THENRETURN
5060 ONRIGOSUB5170,5190,5180,,5160,5120,5200,,,,5140,5130,5150
5070 IFCU<UGTHENCU=CU+DF
5080 IFCU>OGTHENCU=CU-DF
5090 POKECL+CU-UG,6
5100 IFPEEK(CU)<128THENPOKECU,PEEK(CU)+128
5110 GOTO5030
5120 CU=CU-22:RETURN:REM N
5130 CU=CU-21:RETURN:REM NO
5140 CU=CU+01:RETURN:REM O
5150 CU=CU+23:RETURN:REM SO
5160 CU=CU+22:RETURN:REM S
5170 CU=CU+21:RETURN:REM SW
5180 CU=CU-01:RETURN:REM W
5190 CU=CU-23:REM NW
5200 RETURN
10000 POKE37154,0
10010 RI=(PEEK(37152)AND128)+(PEEK(37151)AND(4+8+16))
10020 POKE37154,255
10030 F=PEEK(37151)AND32
10040 RETURN
```

Bild 1.9.1

- Die Joystick-Abfrage wird mit der Cursor-Bewegung gekoppelt. Weil zudem noch die Cursor-Position per Variablen-Abfrage einfach bestimmbar ist, wird dem Programmierer für zahlreiche Anwendungsfälle (auch Spiele) ein nützliches Werkzeug in die Hand gedrückt.

- Der Abfragemechanismus enthält u.a. eine Umrechnungsformel für die Richtungsnummer RI, welche es erlaubt, deren Verwertung zu vereinfachen.

Eine M-PGM-Routine für die Abfrage von einem oder auch zwei angeschlossenen Joysticks finden Sie im Kap. 1.11.

BEDIENUNGSHINWEISE

Das Programm JOYSTICK-CURSOR 1 oder 2 wird mit RUN gestartet. Es wird daraufhin der Cursor in der ersten Bildschirmstelle sichtbar. Der Cursor kann durch entsprechende Richtungsgabe überall hinbewegt werden, auch die diagonalen Richtungen können benutzt werden.

```
0 REM LIES CURSOR PER JOYSTICK
10 GOSUB5000
20 PRINTCU-UG:GOSUB5030:GOTO20
5000 BP=PEEK(648):CL=37888+(BPAND2)*256
5010 POKE37139,PEEK(37139)AND(255-4-8-16-32)
5020 DF=506:CU=BP*256:UG=CU:OG=UG+505
5022 DIMCD(14):FORI=1TO13:READCD(I):NEXT
5024 DATA21,-23,-1,0,22,-22,0,0,0,0,1,-21,23
5030 GOSUB10000:RI=ABS((RI-100)/4)-7
5040 IFPEEK(CU)>127THENPOKECU,PEEK(CU)-128
5050 IFF=0THENRETURN
5060 CU=CU+CD(RI)
5070 IFCU<UGTHENCU=CU+DF
5080 IFCU>OGTHENCU=CU-DF
5090 POKECL+CU-UG,6
5100 IFPEEK(CU)<128THENPOKECU,PEEK(CU)+128
5110 GOTO5030
10000 POKE37154,0
10010 RI=(PEEK(37152)AND128)+(PEEK(37151)AND(4+8+16))
10020 POKE37154,255
10030 F=PEEK(37151)AND32
10040 RETURN
```

Bild 1.9.2

Wird der Bildschirm an einer Seite verlassen, so erscheint er auf der gegenüberliegenden wieder.

Die Feuerknopfbetätigung verursacht in diesen Programmen die Ausgabe der Bildschirmstelle (0...505), wo sich momentan der Cursor befindet (der Wert CU-UG, s. Zeile 20). Dies soll freilich nur ein Beispiel für die Nutzungsmöglichkeiten sein. Dieser Wert kann genauso gut zu anderen Zwecken eingesetzt werden.

1.9 Cursor-Steuerung per Joystick

PROGRAMM-EINZELHEITEN

1.) Name: JOYSTICK-CURSOR 1 und 2

2.) Ausbaustufe: beliebig

3.) Art der PGMe: Hilfsprogramme zur Joystickabfrage

4.) Anzahl der Bytes
JOYSTICK-CURSOR 1: 647
JOYSTICK-CURSOR 2: 517

5.) Benutzte Variablen:

BP	Page-Nummer des 1. Hälfte des Video-Speichers
CL	Anfangsadresse des Farbspeichers
DF	Anzahl der Zeichen auf dem Bildschirm
CU	Absolute Adresse innerhalb des Video-Speichers, wo sich der Cursor befindet
UG	Untere Grenzadresse des Video-Speichers
OG	Obere Grenzadresse des Video-Speichers
RI	Richtungsnummer (s. Bild 1.9.3)
F	Feuerknopf-Flagvariable =0, wenn gedrückt; ansonsten =32
I	Schleifenvariabel (nur in JOYSTICK-CURSOR 2)
CD(14)	beinhaltet die Cursor-Sprungdifferenzen (nur in JOYSTICK-CURSOR 2)

6.) Listings
JOYSTICK-CURSOR 1: Bild 1.9.1
JOYSTICK-CURSOR 2: Bild 1.9.2

7a) Erläuterungen zu JOYSTICK-CURSOR 1:

Zeilen 10 - 20:
Hauptprogramm; in Zeile 10 wird das Unterprogramm ab Zeile 5000 angesprungen, aus dem dann zurückgekehrt wird, wenn der Feuerknopf gedrückt wird. Nach Rückkehr wird in Zeile 20 die Differenz zwischen den absoluten Adressen von Cursorposition und unterer Video-Speicher-Grenze ausgegeben. Letzterer Wert entspricht der Cursor-Positionsnummer (0...505) auf dem Bild-

schirm. Daraufhin wird zur weiteren Joystick-Abfrage das Unterprogramm ab Zeile 5030 angesprungen (der Initialisierungsteil in den Zeilen 5000-5020 muß ausgespart werden). Immer wieder nach dem Drucken des Feuerknopfes kehrt das Programm in die Zeile 20 zuruck.

Richtung	RI	ABS((RI-100)/4)-7
N	152	6
NO	24	12
O	28	11
SO	20	13
S	148	5
SW	132	1
W	140	3
NW	136	2
keine	156	7

Luecken: 4,8,9,10

Bild 1.9.3

Zeile 5000:
Die Video-Speicher-Anfangspage (BP) kann der Speicherzelle #648 entnommen werden. Die Anfangsadresse des Farbspeichers (CL) berechnet sich durch die angegebene Formel.

Zeile 5010:
Die Bits 2,3,4,5 des Datenrichtungsregisters A des VIA #1 mit der Adresse $9113=#37139, welche den Signalen JOY 0 (Norden), JOY 1 (Süden), JOY 2 (Westen), FIRE (Feuerknopf) entsprechen, werden auf 0 gesetzt. D.h. die entsprechenden Eingänge werden auf Eingabe programmiert. Der Eingang für JOY 3 (Osten) ist nicht dem VIA #1 sondern VIA #2 (Bit 7) zugeordnet, kann aber nur unmittelbar vor der Abfrage programmiert werden, weil sonst die Tastatur (zumindest ein Teil davon) blockiert wäre, was das Joystick-Abfrage-Programm etwas einschränken würde.

Zeile 5020:
Initialisierung der Variablen DF, CU, UG, OG

Zeile 5030:
Nach Abfrage der Joystick-Leitungen in Unterprogramm ab Zeile 10000 wird die Richtungsnummer einer Berechnung unterzogen, die eine relativ einfache Zuordnung zur gewählten Richtung erlaubt (s. Zeile 5060 und Bild 1.9.3).

Zeile 5040 - 5050:
In Zeile 5040 wird das invertierte Zeichen, worauf sich der Cursor befindet, in Normaldarstellung gebracht und in Zeile 5050 finden wir den Ausgang des Unterprogramms. Zurückgekehrt wird dann, wenn der Feuerknopf gedrückt wird (F=0).

Zeile 5060:
Nachdem die Richtungsnummer RI der Formel in Zeile 5030 unterzogen worden ist, ist der ON-Verzweigungsbefehl gut nutzbar, weil RI Werte zwischen 1 und 13 angenommen hat (s. Bild 1.9.3). Die Lücken (Werte 4,8,9,10) werden einfach in der ON-Anweisung nicht berücksichtigt und bleiben leer. Je nach Richtung wird ein Unterprogrammsprung zur entsprechenden Cursorverschiebung (Zeilen 5120-5200) vorgenommen.

Zeilen 5070 - 5110:
Danach wird in den Zeilen 5070 und 5080 ein "Wrap around" durchgeführt, d.h. falls der Cursor aufgrund der Berechnung in einer der Zeilen 5120-5190 den Bildschirmbereich verlassen sollte, dann wird er hier wieder "zurückgeholt". Es kann weiterhin passieren, daß der Cursor auf eine Bildschirmstelle ohne Farbe (Vordergrund- = Hintergrundfarbe) "landet". Dadurch, daß in Zeile 5090 die Farbe gesetzt wird, bleibt das Cursor-Blinken immer sichtbar. In Zeile 5100 wird das Zeichen, worauf der Cursor steht, invertiert und in Zeile 5110 wird zum Anfang des Unterprogramms gesprungen.

Zeilen 5120 - 5200:
Je nach Richtung (RI in 5060) wird hier die Cursor-Position verändert.
Zeilen 10000 - 10040:
enthalten die Joystickabfrage. Die Anweisung der Zeile 10000 entspricht im Grunde genommen einer totalen Tastaturblockie-

rung, weil das Datenrichtungsregister für Port B des VIA #2 in Adresse $9122=#37154 vollständig auf Eingabe programmiert wird. Normalerweise steht nämlich Port B auf Ausgabe und Port A auf Eingabe, so daß das Drücken von maximal 63 Tasten erkannt werden kann (8 x 8 abzgl. 1, weil "Keine Taste gedrückt" auch in den Wertebereich der Abfragemöglichkeiten gehört).
In Zeile 10010 werden die Eingänge JOY 0,JOY 1,JOY 2 (die Bits 2,3,4 in Adresse $911F=#37151) sowie JOY 3 (Bit 7 in Adresse $9120=#37152) abgefragt und daraus die Richtungsnummer RI errechnet. Nachdem in Zeile 10020 dann die Tastaturblockierung wieder aufgehoben worden ist, wird der FIRE-Eingang (Bit 5 in Adresse $911F=#37151) in Zeile 10030 abgefragt und das Unterprogramm verlassen.

7b) Erlauterungen zu JOYSTICK-CURSOR 2:

Im Prinzip liegt hier der gleiche Algorithmus wie in JOYSTICK-CURSOR 1 vor, deswegen werde nur auf die Unterschiede gegenuber JOYSTICK-CURSOR 1 eingegangen.

Zeile 5022 - 5024:
Es wird das Feld CD definiert, dessen Elemente die Cursor-Sprungdifferenzen entsprechend der Zeilen 5120-5190 in JOYSTICK-CURSOR 1 sind.

Zeile 5060:
Dadurch, daß sich nunmehr dem neuen Wert RI (1...13) die entsprechende Sprungdifferenz in CD(RI) zuordnen läßt, ist die ehemalige ON-Verzweigung hinfällig geworden. Die Berechnung der neuen Cursor-Position beschränkt sich also auf nur noch eine Programmzeile.

1.10 JOYSTICKABFRAGE-PROGRAMME AUCH FüR 2 JOYSTICKS

Sicherlich haben Sie sich auch schon einmal bei einem schonen Spiel gefragt, bei dem ein Joystick zu bedienen war, ob es denn nicht möglich sei, auch einen zweiten Joystick an den VC-20 anzuschließen. Denn Spiele, in denen Wettkämpfe zwischen zwei Spielpartnern zu gleicher Zeit auszutragen sind, sind vielleicht viel aufregender und interessanter.

1.10 Abfrage von 2 Joysticks

Der Anschluß eines 2. Joystick ist wenig aufwendig: Sie legen sich einfach einen USER-Port-Stecker zu und verbinden (löten) die Kabel des Joystick gemäß Bild 1.10.1 mit den entsprechenden Steckerpins. Das hier beschriebene Prinzip läßt sich übrigens auch auf den Anschluß eines 3. Joystick erweitern.

Der bloße Anschluß eines 2. Joysticks reicht zur Benutzung desselben in einem Programm nicht aus. Dazu ist wieder ein spezielles Abfrageprogramm notwendig.

Anschluss eines 2. Joysticks an den VC20

Pin am Joystick-Stecker	Bedeutung	Pin am USER-Port
1	JOY 0 (Norden)	E
2	JOY 1 (Sueden)	F
3	JOY 2 (Westen)	H
4	JOY 3 (Osten)	L
6	FIRE (Knopf)	J
8	GND (Masse)	A

Bild 1.10.1

Wir stellen Ihnen in diesem Kapitel 2 Möglichkeiten der Joystickabfrage vor:

- Das Basic-Programm DOUBLE-JOYSTICK (s. Bild 1.10.2), welches exakt nach dem gleichen Algorithmus wie CURSOR-JOYSTICK 1 aus Kap. 1.9 arbeitet.

- Das M-PGM JOYSTICK (s. Bild 1.10.4), das eine Abfrage der Joysticks ja wesentlich schneller vornimmt.

Beide Programme sind natürlich auch dann benutzbar, wenn nur ein Joystick angeschlossen ist.

```
0 REM LIES 2 JOYSTICKS
10 GOSUB5000
20 GOSUB5030:GOTO20
5000 REM UP ZUR BEARBEITUNG DER JOYSTICKSTELLUNGEN
5010 POKE37139,PEEK(37139)AND(255-4-8-16-32)
5015 POKE37138,PEEK(37138)AND(255-4-8-16-32-128)
5030 GOSUB10000:RI=ABS((RI-100)/4)-7
5035 GOSUB11000:R2=ABS((R2-100)/4)-7
5050 IFF=0THENPRINT"FEUERN 1":RETURN
5055 IFF2=0THENPRINT"FEUERN 2":RETURN
5060 ONRIGOSUB5170,5190,5180,,5160,5120,5200,,,,5140,5130,5150
5065 ONR2GOSUB5270,5290,5280,,5260,5220,5300,,,,5240,5230,5250
5110 GOTO5030
5120 PRINT"NORD 1":RETURN
5130 PRINT"NORD-OSTEN 1":RETURN
5140 PRINT"OSTEN 1":RETURN
5150 PRINT"SUED-OSTEN 1":RETURN
5160 PRINT"SUEDEN 1":RETURN
5170 PRINT"SUED-WESTEN 1":RETURN
5180 PRINT"WESTEN 1":RETURN
5190 PRINT"NORD-WESTEN 1"
5200 RETURN:REM KEINE RICHTUNG 1
5220 PRINT"NORD 2":RETURN
5230 PRINT"NORD-OSTEN 2":RETURN
5240 PRINT"OSTEN 2":RETURN
5250 PRINT"SUED-OSTEN 2":RETURN
5260 PRINT"SUEDEN 2":RETURN
5270 PRINT"SUED-WESTEN 2":RETURN
5280 PRINT"WESTEN 2":RETURN
5290 PRINT"NORD-WESTEN 2"
5300 RETURN:REM KEINE RICHTUNG 2
10000 POKE37154,0
10010 RI=(PEEK(37152)AND128)+(PEEK(37151)AND(4+8+16))
10020 POKE37154,255
10030 F=PEEK(37151)AND32
10040 RETURN
11000 R2=PEEK(37136)AND(4+8+16+128)
11010 F2=PEEK(37136)AND32
11020 RETURN
```

Bild 1.10.2

1. DAS BASIC-PROGRAMM ZUR JOYSTICKABFRAGE

Dieses Programm ist als Demonstrationsbeispiel anzusehen. Nachdem es mit RUN gestartet wird, wird je nach Richtungsgabe der 2 gleichzeitig benutzbaren Joysticks die entsprechende Richtung, versehen mit "1" für Joystick 1 und mit "2" für Joystick 2, auf dem Bildschirm angezeigt. Wenn die Feuerknopfe gedrückt werden, erscheint "FEUERN 1" bzw. "FEUERN 2". Dort, wo im Programm diese PRINT-Anweisungen vorkommen, können andere Befehle je nach der von Ihnen beabsichtigten Zielsetzung stehen.

1.10 Abfrage von 2 Joysticks

PROGRAMM-EINZELHEITEN

1.) Name: DOUBLE-JOYSTICK

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Joystickabfrage-Programm

4.) Anzahl der Bytes: 916

5.) Benutzte Variablen:

RI	Richtungsnummer des Joysticks 1
R2	" " " 2
F	Feuerknopf-Flagvariable für Joystick 1
F2	" " " " 2

6.) Listing: Bild 1.10.2

7.) Erläuterungen zum Programm:

Prinzipiell können die in Kap. 1.9 aufgeführten Erläuterungen übernommen werden. Es kommen aufgrund der Berücksichtigung des 2. Joysticks folgende Anmerkungen hinzu:

Zeile 5015:
Die Bits 2,3,4,5,7 (JOY 0, JOY 1, JOY 2, FIRE, JOY 3 des 2. Joysticks) des Datenrichtungsregisters für Port B des VIA #1 werden auf Eingabe programmiert.

Zeilen 11000 und 11010:
Die Variable R2 erhält ihre Richtungsnummer entsprechend wie in Zeile 10010. Das gleiche gilt für F2.

Die Auswahl der Eingabeleitungen zum USER-Port für Port B bezgl. des 2. Joystick ist so vorgenommen worden, daß die Bedeutungen der verwendeten Bits die gleichen wie diejenigen für Port A bezgl. des 1. Joysticks sind. Dies hat zur Folge, daß die Behandlung der Richtungsnummern RI und R2 gleich ist. Die Analogien lassen sich unschwer erkennen:

Joystick 1 betreffend	Joystick 2 betreffend
Zeile 5030	Zeile 5035
" 5050	" 5055
" 5060	" 5065
Zeilen 5120-5200	Zeilen 5220-5300
Zeile 10010	Zeile 11000
" 10030	" 11010

2. DAS M-PGM ZUR JOYSTICKABFRAGE

Das Maschinenprogramm JOYSTICK (s. Bild 1.10.4) wird je nach Ihrem Wunsch in einem beliebigen Speicherbereich generiert. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen Überschreiben geschützt und sie können es auch dann aufrufen, wenn Sie beliebige andere Programme im Arbeitsspeicher haben. Weiterhin ist es koppelbar mit von Ihnen frei definierbaren Befehlen (s. Kap. 4.1).

BEDIENUNGSHINWEISE

Das Maschinen-Programm wurde in den im Kapitel 5.2 beschriebenen Basic-Loader (s. Bild 1.10.3) eingebunden. Detailinformation bezgl. diesem kann dort in Erfahrung gebracht werden. Der Basic-Loader wird normal mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das M-PGM generiert werden soll. Bei Eingabe von "E" wird es ans Basic-RAM-Ende gespeichert, wobei eventuell schon andere vorhandene M-PGMe in keinster Weise angetastet werden.

Nachdem Sie die Eingabe der gewünschten Anfangsadresse oder "E" mit RETURN bestätigt haben, wird auf dem Bildschirm der SYS-Befehl angezeigt, mit dem man JOYSTICK aufruft.

Wenn Sie beispielsweise eine 24 KByte-Speichererweiterung besitzen und "E" eingegeben haben, erhalten Sie die Aufruf"Formel":

SYS 32679

1.10 Abfrage von 2 Joysticks

```
10 REM JOYSTICK-ABFRAGE
30 L=82
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA:CH=CH+A
210 POKEI,A:IFPEEK(I)=ATHENNEXT
230 IFCH<>7598THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(146):RETURN
1000 DATA173,19,145,72,41,195,141,19,145,173,34,145,72,41,127,141
1010 DATA34,145,173,31,145,170,41,28,141,62,3,173,32,146,41,128
1020 DATA13,62,3,141,62,3,138,41,32,141,63,3,104,141,34,145
1030 DATA104,141,19,145,234,173,18,145,72,41,67,141,18,145,173,16
1040 DATA145,170,41,156,141,64,3,138,41,32,141,65,3,104,141,18
1050 DATA145,96
```

Bild 1.10.3

```
2000 LDA $9113          Datenrichtungsregister Port A VIA#1
2003 PHA                retten
2004 AND #$C3           PA 5,4,3,2
2006 STA $9113                  auf Input Programmieren
2009 LDA $9122          Datenrichtungsregister Port B VIA#2
200C PHA                retten
200D AND #$7F           PB 7
200F STA $9122                  auf Input Programmieren
2012 LDA $911F          Port A VIA#1 abfragen
2015 TAX                nach X retten
2016 AND #$1C           PA 4,3,2 maskieren
2018 STA $033E          nach $033E = #830
201B LDA $9220          Port B VIA#2
201E AND #$80           PB 7 maskieren
2020 ORA $033E          ueberlagern mit PA 4,3,2 VIA#1
2023 STA $033E          Gesamtergebnis nach $033E
2026 TXA                Port A VIA#1
2027 AND #$20           PA 5 maskieren (FIRE)
2029 STA $033F          nach $033F = #831
202C PLA                    Datenrichtungsregister
202D STA $9122                  wiederherstellen
2030 PLA
2031 STA $9113
2034 NOP
2035 LDA $9112          Datenrichtungsregister Port B VIA#1
2038 PHA                retten
2039 AND #$43           PB 7,5,4,3,2
203B STA $9112                  auf Input Programmieren
203E LDA $9110          Port B VIA#1 abfragen
2041 TAX                nach X retten
2042 AND #$9C           PB 7,4,3,2 maskieren
2044 STA $0340          nach $0340 = #832
2047 TXA                Port B
2048 AND #$20           PB 5 maskieren (FIRE)
204A STA $0341          nach $0341 = #833
204D PLA                    Datenrichtungsregister
204E STA $9112                  wiederherstellen
2051 RTS
```

Bild 1.10.4

Bei der Abfrage der Joysticks in diesem Programm gibt es keine Variablen RI, R2, F, F2. Diese Werte müssen durch PEEK-Befehle den Speicherzellen #830-833 entnommen werden. Dabei entsprechen die Variablen in folgender Weise den Speicherwerten:

Variable	Speicherzelle (#)
RI	830
F	831
R2	832
F2	833

PROGRAMM-EINZELHEITEN

1.) Name: JOYSTICK

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Basic-Loader

4.) Anzahl der Bytes
des Basic-Loaders: 937
des M-PGMs: 82

5.) Benutzte Variablen: s. Kap. 5.2

6.) Listings
des Basic-Loaders: Bild 1.10.3
des M-PGMs: Bild 1.10.4

7.) Erläuterungen zum Basic-Loader werden in Kap. 5.2 gegeben und zum M-PGM implizit im Listing (Bild 1.10.4)

BEISPIEL

Das Programm in Bild 1.10.5 tut im wesentlichen nichts anderes als das Basic-Programm im 1. Teil dieses Kapitels. Daß es um einiges kürzer ist, ist offensichtlich. In diesem Beispiel geschieht die Joystickabfrage über SYS 888 (Zeile 5030), weil ab #888 das M-PGM generiert wurde. Falls es gemäß obigen Generierungsbeispiel am Ende einer 24 KByte-Erweiterung liegt, so müßte in Zeile 5030 der SYS-Befehl lauten: SYS 32679.

Die übrigen Detailinformationen bezgl. diesem Demo-Programm entnehme man den oben getroffenen Ausführungen.

```
0 REM JOYSTICKABFRAGE DEMO
10 GOSUB5000
20 GOSUB5030:GOTO20
5000 REM UP ZUR BEARBEITUNG DER JOYSTICKSTELLUNGEN
5030 SYS888:R1=ABS((PEEK(830)-100)/4)-7
5035 R2=ABS((PEEK(832)-100)/4)-7
5050 IFPEEK(831)=0THENPRINT"F 1":RETURN
5055 IFPEEK(833)=0THENPRINT"F 2":RETURN
5060 ONR1GOSUB5170,5190,5180,,5160,5120,5200,,,,5140,5130,5150
5065 ONR2GOSUB5270,5290,5280,,5260,5220,5300,,,,5240,5230,5250
5110 GOTO5030
5120 PRINT"N 1":RETURN
5130 PRINT"NO 1":RETURN
5140 PRINT"O 1":RETURN
5150 PRINT"SO 1":RETURN
5160 PRINT"S 1":RETURN
5170 PRINT"SW 1":RETURN
5180 PRINT"W 1":RETURN
5190 PRINT"NW 1"
5200 RETURN
5220 PRINT"N 2":RETURN
5230 PRINT"NO 2":RETURN
5240 PRINT"O 2":RETURN
5250 PRINT"SO 2":RETURN
5260 PRINT"S 2":RETURN
5270 PRINT"SW 2":RETURN
5280 PRINT"W 2":RETURN
5290 PRINT"NW 2"
5300 RETURN
```

Bild 1.10.5

1.11 HOCHAUFLÖSENDES ZEICHNEN MIT DEM JOYSTICK

Das ziemlich kurze Programm (556 Bytes) in Bild 1.11.1 verwendet die Joystickabfrage zum Zeichnen eines Bildes in hochauflosender Grafik. Im Kapitel 6.10 erfahren Sie noch weitere Einzelheiten über die hochauflösende Graphik.

Dieses Programm funktioniert nur im GV-Modus. Ein VC-20 mit Speichererweiterung läßt sich schnell über die im Kap. 6.3 besprochenen Methoden in diesen Modus "umschalten".

Nachdem RUN eingegeben worden ist, tut sich erst einmal einiges auf dem Bildschirm. Auf schwarzem Hintergrund wird ein weißer Rahmen aufgebaut und in der Mitte erscheint ein weißer Punkt in der Größe eines einzelnen Pixels, gewissermaßen ein Mini-Cursor. Mit dem Joystick können nun Linien je nach der von Ihnen vorgegebenen Richtung gezeichnet werden. Wenn der "Cursor" ohne zu zeichnen bewegt werden soll, wird einfach auf den Feuerknopf gedrückt. Das Auswischen des Bildes geschieht durch Drücken auf die SPACE-Taste.

```
0 REM ZEICHNEN MIT JOYSTICK
10 POKE36879,8:POKE36867,21:POKE36864,17
20 POKE36865,45:POKE36866,144:POKE36869,253
30 PRINTCHR$(147):FORA=1TO16:FORN=1TO10
40 POKE7663+16*N+A,N+10*A-11:NEXTN,A
50 FORA=5120TO7679:POKEA,0:NEXT:POKE37154,127:X=64:Y=80
60 FORA=5120TO7679STEP160:POKEA,255:POKEA+159,255:NEXT
70 FORA=5121TO5278:POKEA,128:POKEA+2400,1:NEXT
80 X2=63:Y2=79
90 A=PEEK(37137):X=X+((AAND16)=0):Y=Y+((AAND4)=0)
100 Y=Y-((AAND8)=0):X=X-((PEEK(37152)AND128)=0)
110 IFPEEK(203)AND63THENRUN
120 IFX2=XANDY2=YTHEN90
130 F=-((AAND32)=0):IFF=0THEN150
140 Y1=Y2+5120+160*INT(X2/8):POKEY1,PEEK(Y1)ANDNOT2↑(7-X2AND7)
150 Y1=Y+5120+160*INT(X/8):POKEY1,PEEK(Y1)OR2↑(7-XAND7)
160 X2=X:Y2=Y:GOTO90
```

Bild 1.11.1

2

Basic-Programmierhilfen

2.1 SPEICHERORGANISATION BEI EINEM BASIC-PROGRAMM

Für manche Anwendungen wie Progammanipulationen, "Verstecken" eines Maschinenprogramms in einem Basic-Programm, Einbringen von Schutzmechanismen etc. ist die Kenntnis darüber unbedingt erforderlich, wie ein Basic-Programm im VC-20-Arbeitsspeicher vom Betriebssystem organisiert ist.

ANFANG UND ENDE DES BASIC-ARBEITSSPEICHERS

Der Basic-Arbeitsspeicher wird durch 2 Vektoren begrenzt:

```
Speicherzellen #43,44 = $2B,2C:     Basic-Anfang (BA)
"              #55,56 = $37,38:     Basic-Ende (BE)
```

Nach Einschalten des Gerats oder RESET werden diese Vektoren auf die in Bild 2.1.1 aufgefuhrten Anfangswerte gesetzt, falls dies nicht durch ein Modulprogramm mit Autostart (s. Kap. 6.5) verhindert wird.

Inhalt in:	#44 = $2C		#56 = $38	
	#	$	#	$
GV	16	10	30	1E
3K-V	4	04	30	1E
8K-V	18	12	64	40
16K-V	18	12	96	60
24K-V	18	12	128	80

Bild 2.1.1

Die Inhalt in #43=$2B ist in allen Fällen 1 und in #55=$37 der Wert 0.

Überprufen Sie die o.g. Werte speziell in Ihrem Gerät durch:

```
PRINT PEEK(43);PEEK(44);PEEK(55);PEEK(56)   (Return)
```

Da die Vektoren im LOW/HIGH-Format gespeichert sind, ergeben sich für die Vektoren BA und BE die absoluten Adressen zu den Werten gemäß Bild 2.1.2.

Zu BA muß ergänzend gesagt werden, daß es zum ordentlichen Funktionieren eines Basic-PGMs unbedingt erforderlich ist, daß die Speicherzelle unmittelbar unter BA den Wert 0 enthalt. Folgerichtig befindet sich in den Zellen #4096 bzw. 1024 bzw. 4608 der Wert 0. Diese 0 kann als Bestandteil des Basic-Bereichs aufgefaßt werden, so daß BA in Wirklichkeit auf die 2. Speicherzelle des Basic-Arbeitsspeichers zeigt.

Bei BE können wir eine ähnliche Aussage wie bei BA treffen. Der Vektor #55,56 zeigt nämlich nicht auf exakt die letzte Speicherzelle, sondern auf die darüber. Diese gehört bereits in einen anderen Bereich. Im Fall GV zählt die Adresse #7680 zum Video-Speicher, in der 24K-V die Adresse #32768 zum Zeichengenerator.

Die Vektoren BA und BE lassen sich durch POKE-Befehle je nach Anwendungsfall nach Belieben verändern, nur sollte man darauf achten, daß BE > BA ist, und ein durchgehender RAM-Bereich zwischen BA und BE vorhanden ist. Wenn z.B. der Basic-Arbeitsspeicher zwischen den Adressen BA = #5001 = 137+256*19 und BE = #6000 = 112+256*23 festgelegt werden soll (die LOW/HIGH-Anteile lassen sich mühelos mit dem Programm VC-CONVERT in Kap. 5.3 berechnen), dann ist die folgende Eingabe im Direkt-Modus nötig:

```
POKE 43,137:POKE 44,19:POKE 55,112:POKE 56,23:POKE 5000,0:NEW
```

	BA		BE	
	#	$	#	$
GV	4097	1001	7680	1E00
3K-V	1025	0401	7680	1E00
8K-V	4609	1201	16384	4000
16K-V	4609	1201	24576	6000
24K-V	4609	1201	32768	8000

Bild 2.1.2

Das POKE 5000,0 ist erforderlich aufgrund der oben getroffenen Aussage, daß am Basic-Anfang eine 0 stehen muß, und diese nicht automatisch (ausgenommen beim Einschalten oder RESET) erzeugt wird. Das NEW sorgt dafur, daß die anderen Basic-Vektoren, auf die wir gleich eingehen werden, automatisch den neuen Verhältnissen angepaßt werden.

Aufgrund der bisherigen Ausführungen kann übrigens der Schluß gezogen werden, daß mit dieser Arbeisspeicherfestlegungsmethode mehrere Basic-Programme, welche in Hinblick auf die verwendeten Zeilennummern voneinander völlig unabhängig sein dürfen, in den insgesamt verfügbaren RAM untergebracht werden können. Das "Umschalten" geschieht jeweils durch Befehlszeilen gemaß obigem Beispiel.

UNTERTEILUNG DES BASIC-ARBEITSSPEICHERS

Nachdem wir im ersten Abschnitt dieses Kapitels gelernt haben, wie der Anfang und das Ende des Arbeitsspeichers nach Belieben festgelegt werden kann, wenden wir uns nun der Frage zu, wie innerhalb dieser Grenzen Variablen, Felder und Daten verwaltet werden.

5 Vektoren haben die Aufgabe, die einzelnen Bereiche des Arbeitsspeichers voneinander abzugrenzen:

Speicherzellen	#45,46 = $2D,2E:	Variablen-Anfang (VA)
"	#47,48 = $2F,30:	Felder-Anfang (FA)
"	#49,50 = $31,32:	Felder-Ende (FE)
"	#51,52 = $33,34:	String-Anfang (SA) des zuletzt gespeicherten Strings bzw. Anfang der String-Daten
"	#53,54 = $35,36:	String-Ende (SE) des zuletzt gespeicherten Strings

Zwischen BA und VA-1 liegt das Basic-Programm. Ab VA sind die Variablen und Funktionen gespeichert. Wie diese abgespeichert sind, erfahren Sie in Kap. 4.11 "DUMP per Basic". Die obere Grenze für die Variablen und Funktionen ist FA-1. Von FA bis FE-1 reicht der Speicherplatz für die Felder.

An sich sind die Werte der Variablen und Felder in den eben genannten Bereichen integriert. Dies gilt allerdings nicht fur die String-Variablen und -Felder. Das liegt daran, daß bei Strings nicht von vornherein klar ist, aus wieviel einzelnen Zeichen die eine oder andere Variable bestehen wird. In größeren Rechenanlagen kann man es sich leisten, fur Strings Speicherbereiche mit vorgegebener Lange zu reservieren. Beim VC-20-Basic, wie beim Basic vieler anderer Heimrechner auch, ist man zwecks optimaler Speicherausnutzung jedoch den Weg gegangen, daß die String-Daten ans Basic-Ende gelegt werden.

Der Vektor SA hat zwei Aufgaben: normalerweise zeigt er auf das 1. Zeichen der zuletzt abgespeicherten String-Datenkette, die im Laufe des Programms oder auch im Direkt-Modus einer String-Variablen bzw. einem String-Feldelement zugewiesen worden ist. Beim Abspeichern kann es jedoch vorkommen, daß das Betriebssystem gezwungen wird, samtliche "Garbage"-Daten - String-Daten also, welche irgendwann vorher einmal einer Variablen oder einem Feldelement zugewiesen waren, nun aber nicht mehr - herauszuwerfen (die sogannte "Garbage-Collection"). Dies geschieht dann, wenn sonst ohnehin kein Platz mehr zum Abspeichern vorhanden ware oder aber, wenn die Funktion FRE aufgerufen wird. Tritt einer dieser beiden Falle auf, dann zeigt der Vektor SA grundsatzlich auf den Anfang (also auf das untere Ende) samtlicher vorhandener und zugewiesener String-Daten. Es ist daraus unschwer ableitbar, daß das Resultat der Funktion FRE genau der Differenz der Vektoren SA und FE entsprechen muß.

```
10 GOSUB1000
20 INPUTA:GOSUB1000
30 INPUTE$:GOSUB1000
40 INPUTF$:GOSUB1000
50 DIMB(3):INPUTB(0):GOSUB1000
60 PRINT:PRINT"CLR":CLR:GOSUB1000
70 END
1000 PRINT"BA=";PEEK(43)+256*PEEK(44)
1010 PRINT"VA=";PEEK(45)+256*PEEK(46)
1020 PRINT"FA=";PEEK(47)+256*PEEK(48)
1030 PRINT"FE=";PEEK(49)+256*PEEK(50)
1040 PRINT"SA=";PEEK(51)+256*PEEK(52)
1050 PRINT"SE=";PEEK(53)+256*PEEK(54)
1060 PRINT"ME=";PEEK(55)+256*PEEK(56)
1070 RETURN
```

Bild 2.1.3

SE zeigt normalerweise auf die Speicherzelle, welche hinter der zuletzt abgespeicherten String-Datenkette liegt. Der Wert von SE hat keine Bedeutung, wenn eine Garbage-Collection eingetreten ist.

Anhand des Beispiel-Programms in Bild 2.1.3 sollen die Werte der o.g. Vektoren und deren Veranderungen erlautert werden. Wie gehen davon aus, daß dieses Programm in einem VC-20 in GV geladen wird (Umschalten in GV: s. Kap. 6.3). Wird das Programm mit RUN gestartet, so wird als erstes das Unterprogramm ab Zeile 1000 aufgerufen, worin die Werte sämtlicher oben besprochenen Vektoren ausgegeben werden (Bild 2.1.4a). Die Länge des Programms ist einfach durch Differenzbildung von VA und BA ermittelbar; es ist also 297 Bytes lang. Wie Sie durch Vergleich mit den Vektor-Werten für VA und BA nach den weiteren Eingaben erkennen können, bleiben diese konstant, die Programmlänge bleibt demnach unverändert. Das muß nicht notwendigerweise immer so sein. Es gibt auch Programme, in denen per POKE-Befehl der Vektor VA verändert wird, z.B. wenn Programmteile von Diskette nachgeladen werden sollen etc.

BA= 4097	BA= 4097	BA= 4097
VA= 4394	VA= 4394	VA= 4394
FA= 4394	FA= 4401	FA= 4408
FE= 4394	FE= 4401	FE= 4408
SA= 7680	SA= 7680	SA= 7677
SE= xxxx	SE= xxxx	SE= 7680
BE= 7680	BE= 7680	BE= 7680
Bild 2.1.4a	Bild 2.1.4b	Bild 2.1.4c

Die Eingabe von A in Zeile 20 bewirkt, daß für die Variable A sieben Bytes von VA an aufwärts reserviert werden. Deswegen verschieben sich FA und FE um 7 Bytes nach oben (Bild 2.1.4b).
In Zeile 30 wird der String E$ eingegeben, z.B. "ABC". Zur Verwaltung von E$ muß zwischen FA und VA wieder Platz für 7 Bytes geschaffen werden (Bild 2.1.4c). Außdem wird "ABC" ans Basic-Ende gelegt. SA wird hierbei auf die Speicherzelle gerichtet, die das 1. Zeichen, also "A" enthält. SE hatte in den ersten beiden Fällen keine Bedeutung (deswegen xxxx in den ersten Bildern), jetzt aber zeigt SE auf die Speicherzelle unmittelbar hinter "C".

In Zeile 40 wird abermals ein String eingegeben, die Variable F$; wir nehmen z.B. an: "123". Wieder muß zwischen VA und FA Platz für 7 Bytes eingerichtet werden. Außerdem werden SA und SE aufgrund dieser Eingabe entsprechend ausgerichtet (Bild 2.1.4d).

BA= 4097	BA= 4097	BA= 4097
VA= 4394	VA= 4394	VA= 4394
FA= 4415	FA= 4415	FA= 4394
FE= 4415	FE= 4442	FE= 4394
SA= 7674	SA= 7674	SA= 7680
SE= 7677	SE= 7677	SE= xxxx
BE= 7680	BE= 7680	BE= 7680
Bild 2.1.4d	Bild 2.1.4e	Bild 2.1.4f

Bisher hatten FA und FE die gleichen Werte, weil im Programm kein Feld benutzt worden ist. In Zeile 50 jedoch wird das eindimensionale Feld B definiert, welches 4 Elemente (0 eingeschlossen) aufweisen soll. Aus der Differenz von FE und FA können wir ableiten, daß für das Feld B 27 Bytes reserviert wurden (Bild 2.1.4e). Diese Zahl wird durch die allegemeine Formel bestätigt:

Benötigter Speicherplatz für Felder = 5 + 2*D + E*T

mit: D = Dimension (z.B.: B(3) ist eindimensional,
C(4,5) ist zweidimensional etc.)
E = Insgesamte Anzahl der Feldelemente
(z.B.: B(3) enthält 4 Elemente, C(4,5)
besteht aus 30 Elementen)
T = 5 für Gleitkomma-Felder
= 3 " String-Felder
= 2 " Integer-Felder

Der momentane Zustand im Arbeitsspeicher in Verbindung mit den Werten der hier besprochenen Vektoren wird zusätzlich mittels Bild 2.1.5 veranschaulicht.

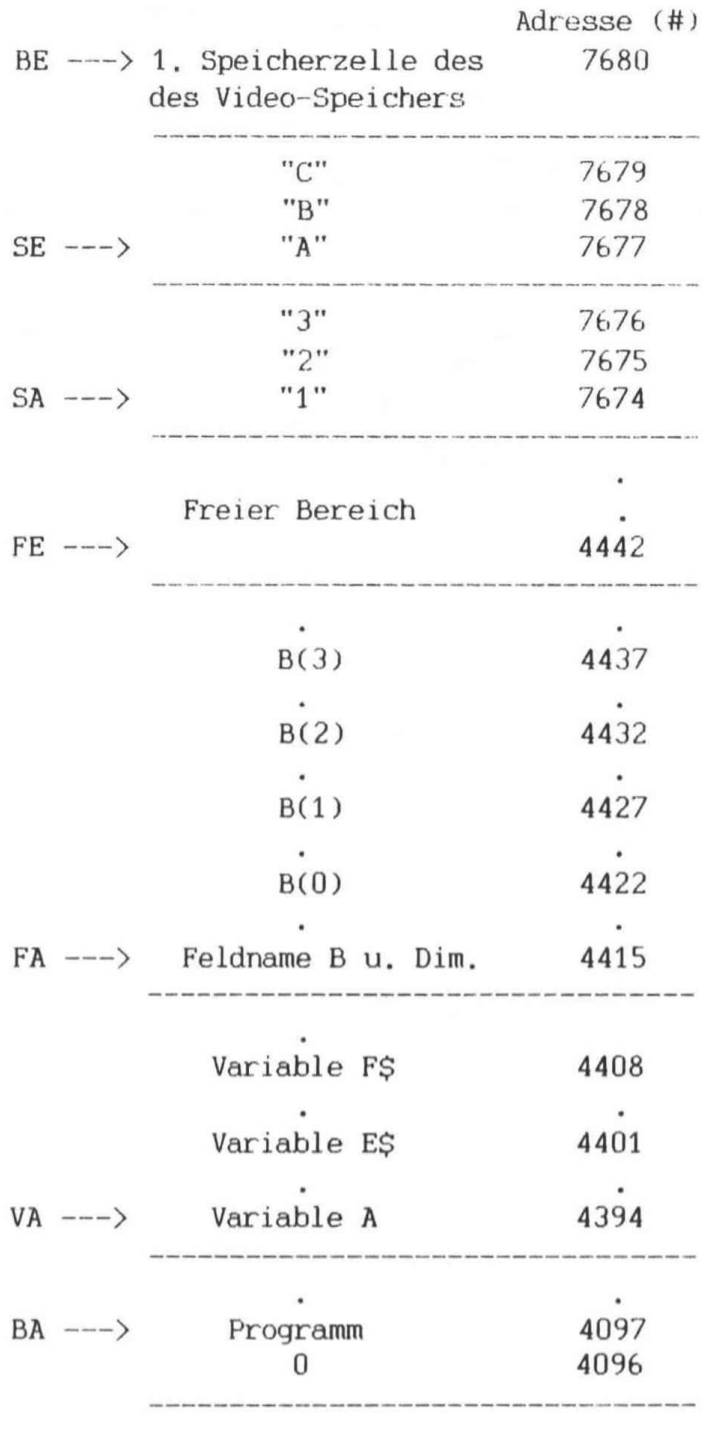

Adresse (#)
BE ---> 1. Speicherzelle des des Video-Speichers
7680
"C"
7679
"B"
7678
SE ---> "A"
7677
"3"
7676
"2"
7675
SA ---> "1"
7674
Freier Bereich
FE ---> 4442
B(3)
4437
B(2)
4432
B(1)
4427
B(0)
4422
FA ---> Feldname B u. Dim.
4415
Variable F$
4408
Variable E$
4401
VA ---> Variable A
4394
BA ---> Programm
4097
0
4096

Bild 2.1.5

In Zeile 60 des Programms wird die Funktion CLR benutzt. In Bild 2.1.4f ist erkennbar, daß nach CLR sämtlicher Speicherplatz für Variablen und Felder wieder freigegeben wurden. Es liegen sodann die gleichen Verhältnisse wie bei Programmeinstieg vor.

Wie Variablen gespeichert werden, haben wir in Kap. 4.11 erfahren. Es steht noch die Antwort auf folgende Frage aus:

WIE WERDEN FELDER GESPEICHERT ?

Betrachten wir zu Anfang der Einfachheit halber eindimensionale Felder, diejenigen also, denen bei der Definition per DIM-Anweisung nur eine Zahl für die Anzahl der Feldelemente zugewiesen wird. Z.B. wird in der Zeile:

```
10 DIM AB(1000)
```

ein eindimensionales Gleitkomma-Feld definiert, welches 1001 Feldelemente aufweist (das Feldelement AB(0) gehört ja dazu). Wir wissen zwar aus der oben angegebenen Formel bereits, daß für das Feld AB 5012 Bytes Speicherplatz gebraucht werden, jedoch nicht, welche Bedeutung die einzelnen Bytes haben. Bild 2.11.6 veranschaulicht das Speicherungsprinzip für Gleitkomma-Felder. Für NZ1,NZ2 sowie die Werte gelten die Regeln aus Kap. 4.11 (Funktionsdefinitionen gibt es bei Feldern allerdings nicht).

Byte	1	2	3	4	5	6	7
Inhalt	NZ1	NZ2	IS(L)	IS(H)	DIM	FZ(H)	FZ(L)

Byte	8	9	10	11	12	13	14
Inhalt	Wert des Feldelements (0)					Wert des	

Byte	15	16	17	18	19	20	21
Inhalt	Feldelements (1)		!		...		

Bild 2.1.6: Eindimensionales Gleitkomma-Feld

Mit:	IS(L),IS(H)	Speicherplatz, der für das Feld insgesamt benötigt wird (s. obige Formel), im Beispiel: 148, 19
	DIM	Anzahl der Dimensionen, im Beisp.: 1
	FZ(H),FZ(L)	Feldelement-Anzahl im Beisp.: 3, 233

Das eindimensionale Integer-Feld unterscheidet sich vom Gleitkomma-Feld dadurch, daß zur Abspeicherung des Werts jedes Feldelements nur 2 Bytes anstatt 5 Bytes erforderlich sind. Beim String-Feld werden jedem Feldelement-Wert 3 Bytes zugewiesen, die natürlich nicht die Daten enthalten, sondern die Länge und die Anfangsadresse der Daten. Mehr Details dazu erfahren Sie im Kap. 4.11.

Byte	1	2	3	4	5	6	7
Inhalt	NZ1	NZ2	IS(L)	IS(H)	DIM	F3(H)	F3(L)
Byte	8	9	10	11	12	13	14
Inhalt	F2(H)	F2(L)	F1(H)	F1(L)	Wert (0,0,0)		Wert
Byte	15	16	17	18	19	20	21
Inhalt	(1,0,0)	Wert (0,1,0)		Wert (1,1,0)		Wert (0,2,0)	...

Mit:	IS(L),IS(H)	Insgesamter Speicherplatz, der für das Feld benötigt wird (s. obige Formel), im Beispiel: 59, 0
	DIM	Anzahl der Dimensionen, im Beisp.: 3
	F3(H),F3(L)	Feldelement-Anzahl der 3. Dimension, im Beisp.: 0, 4
	F2(H),F2(L)	Feldelement-Anzahl der 2. Dimension, im Beisp.: 0, 3
	F1(H),F1(L)	Feldelement-Anzahl der 1. Dimension, im Beisp.: 0, 2

Bild 2.1.7: Dreidimensionales Integer-Feld

Bei den mehrdimensionalen Feldern gehen wir wieder von einem Beispiel aus. Die Anweisung:

```
20 DIM XY%(1,2,3)
```

definiert ein dreidimensionales Integer-Feld des Namens XY% und enthält (1+1)*(2+1)*(3+1)=24 Feldelemente. Gemäß obiger Formel werden zur Abspeicherung des Feldes XY% insgesamt 59 Bytes benötigt. Das Speicherungsprinzip hierbei ist dem Bild 2.1.7 zu entnehmen. Was hier für 3 Dimensionen gezeigt ist, läßt sich auf Dimensionen höherer Ordnung übertragen. Nach der Dimensionszahl im 5. Byte werden hintereinander im HIGH/LOW-Format die Feldelement-Anzahl der einzelnen Dimensionen (bei der letzten wird angefangen) abgelegt, wonach dann die Werte der Feldelemente folgen. Hochgezählt wird die 1. Dimension, dann die 2. etc. Für das Format der Werte gelten die in Kap. 4.11 beschriebenen Regeln.

PROGRAMM-STRUKTUR

Nachdem das Drumherum von Programmen in genügender Weise erläutert sein dürfte, wenden wir uns zum Schluß der Frage zu, wie denn ein Programm selbst organisiert ist. Zunächst einmal muß das Betriebssystem wissen, wo das zu bearbeitende Basic-PGM beginnt. Der Vektor #43,44 gibt darüber Auskunft, was ja am Anfang dieses Kapitels schon ausreichend erläutert wurde. Dieser zeigt demnach auf die 1. Programmzeile.

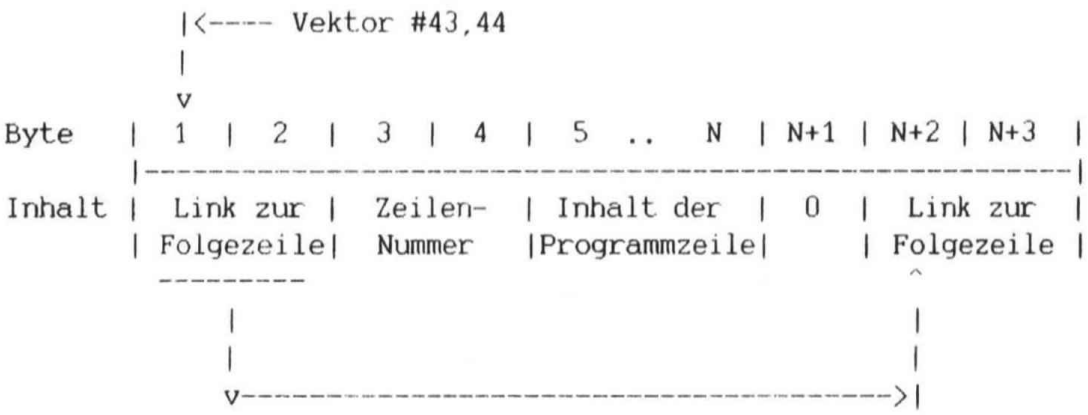


Bild 2.1.8: 1. Basic-Programmzeile

In Bild 2.1.8 ist das Prinzip der Zeilenverkettung erkennbar. Die ersten 2 Bytes (im Bild die Bytes 1 und 2) enhalten im gewohnten LOW/HIGH-Format die Adresse des 1. Bytes der Folgezeile (Byte N+2). Diese Adresse wird als Link bezeichnet. Auf den Link folgt die von Ihnen gewählte Zeilennummer, ebenfalls im LOW/HIGH-Format. Es sind dafur grundsatzlich 2 Bytes vorgesehen. Deswegen ist die desofteren getroffene Behauptung, man konne Speicherplatz durch Benutzung kleiner Programmzeilennummern sparen, nur teilweise richtig. Man spart namlich nur dann etwas, wenn die Zeilennummern im Zeileninhalt vorkommen, z.B. in den Anweisungen mit GOTO, GOSUB, THEN und ON. Dort werden diese Nummern als ASCII-Zeichenketten gespeichert, jede Ziffer verbraucht dabei ein Byte. Nicht jedes Zeichen des Inhalts einer Programmzeile wird im ASCII-Code abgespeichert. Die Basic-Befehle, -Funktionen sowie die arithmetischen und Booleschen Operatoren werden als sogenannter Token, bestehend aus 1 Byte, abgelegt.

END	80	ON	91	NEW	A2	RND	BB
FOR	81	WAIT	92	TAB(	A3	LOG	BC
NEXT	82	LOAD	93	TO	A4	EXP	BD
DATA	83	SAVE	94	FN	A5	COS	BE
INPUT#	84	VERIFY	95	SPC(	A6	SIN	BF
INPUT	85	DEF	96	THEN	A7	TAN	C0
DIM	86	POKE	97	NOT	A8	ATN	C1
READ	87	PRINT#	98	STEP	A9	PEEK	C2
LET	88	PRINT	99	AND	AF	LEN	C3
GOTO	89	CONT	9A	OR	B0	STR$	C4
RUN	8A	LIST	9B	SGN	B4	VAL	C5
IF	8B	CLR	9C	INT	B5	ASC	C6
RESTORE	8C	CMD	9D	ABS	B6	CHR$	C7
GOSUB	8D	SYS	9E	USR	B7	LEFT$	C8
RETURN	8E	OPEN	9F	FRE	B8	RIGHT$	C9
REM	8F	CLOSE	A0	POS	B9	MID$	CA
STOP	90	GET	A1	SQR	BA	GO	CB
+	AA	-	AB	*	AC	/	AD
^	AE	>	B1	=	B2	<	B3

Bild 2.1.9: Basic-Token (in HEX)

In Bild 2.1.9 sind samtliche Basic-Token in HEX aufgeführt. Dort, wo also in einer Programmzeile eines dieser Zeichenzusammensetzungen (außer innerhalb von "") auftaucht, bzw. die im Handbuch auf S. 133 angegebenen Kurzel, werden diese durch den entsprechenden 1-Byte-Token ersetzt.
Zur Kennzeichnung des Programmzeilenendes dient ein Byte mit dem Wert 0 (Byte N+1 in Bild 2.1.8). Ist die letzte Zeile erreicht, so zeigt zwar deren Link auf den scheinbaren Anfang der Folgezeile. Dieser Anfang jedoch besteht aus 2 Nullen. Das Betriebssystem interpretiert darin das Ende des Programms. Bild 2.1.10 gibt diesen Sachverhalt wieder.

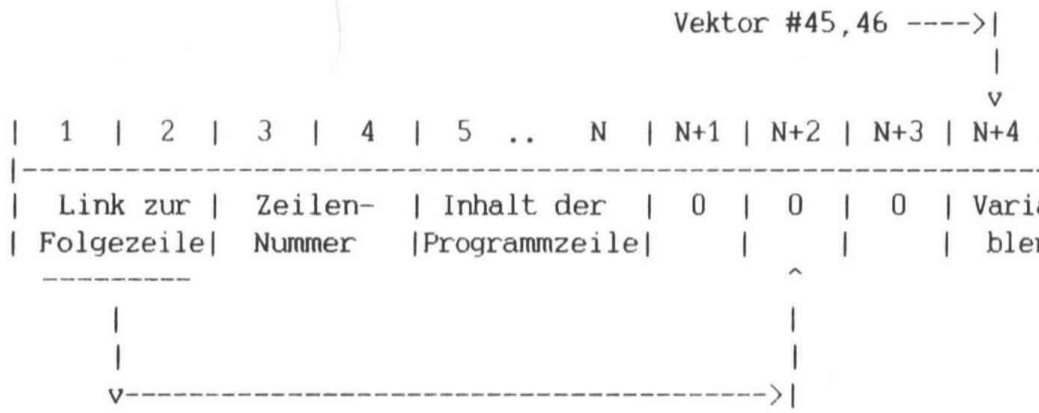


Bild 2.1.10: Letzte Basic-Programmzeile

Zur erganzenden Verdeutlichung wählen wir folgendes Programmbeispiel:

```
2 PRINT22
4 PRINT44
6 PRINT66
8 PRINT88
```

In einem VC-20 in GV wäre das Programm in der in Bild 2.1.11 dargestellten Weise ab Adresse $1001 gespeichert (Inhalte in HEX). In $1001,1002 befindet sich der Vektor $1009, der auf die 2. Programmzeile zeigt. Darauf folgt die Zeilennummer 2, der Token $99 für den Befehl PRINT und schließlich die ASCII-Codes von "22", also $32,32. Bis zum Programmende ändert sich prinzipiell nichts.

Man kann das Betriebssystem ein wenig austricksen, was den Link-Mechanismus angeht, indem die Links "verbogen" werden. Wir nehmen beispielsweise mit den Inhalten der Zellen $1001,1009,1011 einen zyklischen Tausch vor, so daß diese hinterher die Werte $11,19,09 enthalten. Der Link der 1. Zeile zeigt somit auf die 3. Zeile, der Link der 3. Zeile auf die 2. Zeile und der Link der 2. Zeile auf die 4. Zeile. Tatsächlich ergibt sich als Resultat dieser Aktion nach LIST die Ausgabe von:

```
2 PRINT22
6 PRINT66
4 PRINT44
8 PRINT88
```

Allerdings bei RUN laßt sich das Betriebssystem in dieser Weise nicht betrügen. Die Zahlen 22,44,66,88 werden in der richtigen Reihenfolge am Bildschirm angezeigt.

```
Adresse: 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
         00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
         -----------------------------------------------
Inhalt:  00 09 10 02 00 99 32 32 00 11 10 04 00 99 34 34

Adresse: 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
         10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22
         -------------------------------------------------------
Inhalt:  00 19 10 06 00 99 36 36 00 21 10 08 00 99 38 38 00 00 00
```

Bild 2.1.11

Zum Abschluß werde noch auf die Tatsache hingewiesen, daß (s. Bild 2.1.10) vom Betriebssystem aus der Vektor #45,46, also der Variablenanfang, auf die 1. Speicherzelle hinter den 2 Programmendenullen ausgerichtet wird. Jedoch laßt sich dieser von Ihnen in der Weise verändern, daß Sie durch dessen Erhöhung das Programm "aufblahen". Dies wirkt sich vor allem beim SAVEn aus, weil die SAVE-Routine alles das abspeichert, was zwischen den Vektoren VA und BA liegt. Auf diese Weise können die während eines Programms anfallenden Daten oder hinzugefügte Maschinenprogramme mitgeSAVEd werden, ohne daß dazu ein Filehandling nötig ist.

EINE PROGRAMMZEILE MIT MEHR ALS 88 ZEICHEN ?

Diese Frage wurde uns schon des öfteren gestellt. Was hat es damit auf sich? Es gibt zahlreiche Programmierer, die dadurch Bytes für ein Programm einsparen wollen, daß sie mehrere Anweisungen in eine Programmzeile packen und durch ein ":" voneinander trennen. Diese Methode ist allen von Ihnen bekannt und es versteht sich von selbst, daß dabei 4 Bytes pro zusätzlicher Anweisung eingespart werden (der Link sowie die Zeilennummer von je 2 Bytes entfallen ja). Was jedoch nicht allen von Ihnen geläufig sein dürfte, ist die Tatsache, daß bei dieser Byte-Sparmethode in Verbindung mit der Verwendung der Anweisungs-Kürzel gemäß VC-20-Handbuch S. 133 zwar die eingetippte Programmzeile nicht länger als 88 Zeichen (= 4 Zeilen) wird, aber bei LIST-Aufruf diese sich plötzlich als eine Zeile mit mehr als 4 Zeilen entpuppt.

Führen wir diesen Effekt einfach mit folgender Eingabe vor, die Sie bitte nachvollziehen:

```
5?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:
?:?:?:?:?:?  (Return)
```

Diese Programmzeile besteht aus 86 eingetippten Zeichen, erreicht somit noch nicht einmal die erlaubte Grenze von 88 Zeichen. Sie werden sich aber wundern, wenn sie sich diese Zeile mittels LIST anschauen. Mit einer Länge von fast 12 Zeilen hat diese sich ganz schön aufgebläht. Allerdings besteht sie jetzt nur scheinbar aus 259 Zeichen. Denn die Anweisungen (in unserem Beispiel lauter PRINTs) werden als 1-Byte-Token (s.o.) abgespeichert. Diese Token werden bei LIST in die entsprechenden aus mehr als 1 Zeichen bestehenden Anweisungen umgewandelt.

Diesen Effekt zu verstehen kann für Sie dann von Bedeutung sein, wenn Ihnen ein Programm-Listing in die Hände fällt, welches Programmzeilen von mehr als 88 Zeichen Länge enthält. So kam ja auch die Frage von so manchen auf: "Wie soll ich denn das eintippen? Das paßt ja gar nicht in die 4 Zeilen hinein." Um dieses Problem zu lösen, wenden Sie einfach die gleiche Methode an wie der Programmierer selbst, indem Sie ebenfalls die Anweisungs-Kürzel beim Eintippen benutzen, also z.B. anstatt "PRINT" das "?" eingeben.

2.2 VARIABLEN-TABELLE

Haben Sie schon gewußt, daß der insgesamt verfügbare Namensraum im VC-20-Basic aus 6731 Namen besteht? Dabei sagt man doch desöfteren, man sei durch die Basic-Vorschrift bezgl. der Namensgebung für die Variablen und Felder arg eingeschränkt. Die Zahl 6731 setzt sich zusammen aus:

- 962 Namen für Funktionen
- 1924 Namen für Integer-Variablen und -Felder
- 1923 Namen für String-Variablen und -Felder
- 1922 Namen für Gleitkomma-Variablen und -Felder

Während der Erstellung eines Programms wird es Ihnen vielleicht auch manchmal so ergangen sein, daß Sie nicht mehr genau wußten, ob Sie die eine oder andere Variable bereits benutzt haben. Aus diesem Grund ist es zweckmäßig, sich per Tabelle während des Programmerstellungsprozesses (auch hinterher natürlich) ständig darüber im Klaren sein zu können, welche Variable bzw. Funktion bzw. welches Feld mit welcher Bedeutung gebraucht werden.

Die Tabelle in Bild 2.2.1 soll einen Überblick über die Namensvergabe schaffen. Am besten kopieren Sie sich dieses Bild und die Folgebilder mehrfach, damit Sie bei Bedarf immer welche parat haben. Brauchen Sie z.B. die Variable MR$ in Ihrem Programm, so suchen Sie das 1. Namenszeichen "M" in der obersten Reihe. Von dort aus fahren Sie in der M-Spalte so lange nach unten, bis Sie zum Kreuzungspunkt der R-Reihe gelangen. Dort tragen Sie beispielsweise eine laufende Nummer ein, auf die Sie in den ersten Spalten der Folgebilder 2.2.2 und 2.2.3 bezugnehmen, indem Sie dort weitere Angaben über Typ, Dimension, Bedeutung, Name etc. treffen können. Im Fall von MR$ tragen Sie z.B. eine 1 in das entsprechende Feld in Bild 2.2.1 ein. Die 1 übertragen Sie dann vorne oben ins Bild 2.2.2 und machen ein Kreuz in die 4. Spalte ($). Diese Tabelle kann zu jeder Zeit während der Programmierung Aufschluß über den benötigten Speicherplatz geben.

In den Feldern für TI und ST befinden sich Fragezeichen, weil TI nur als TI% und ST nur als ST% und ST$ verfügbar sind. Für Namen mit nur einem Namenszeichen ist die 1. Reihe des Bildes 2.2.1 vorgesehen.

2.2 Variablen-Tabelle

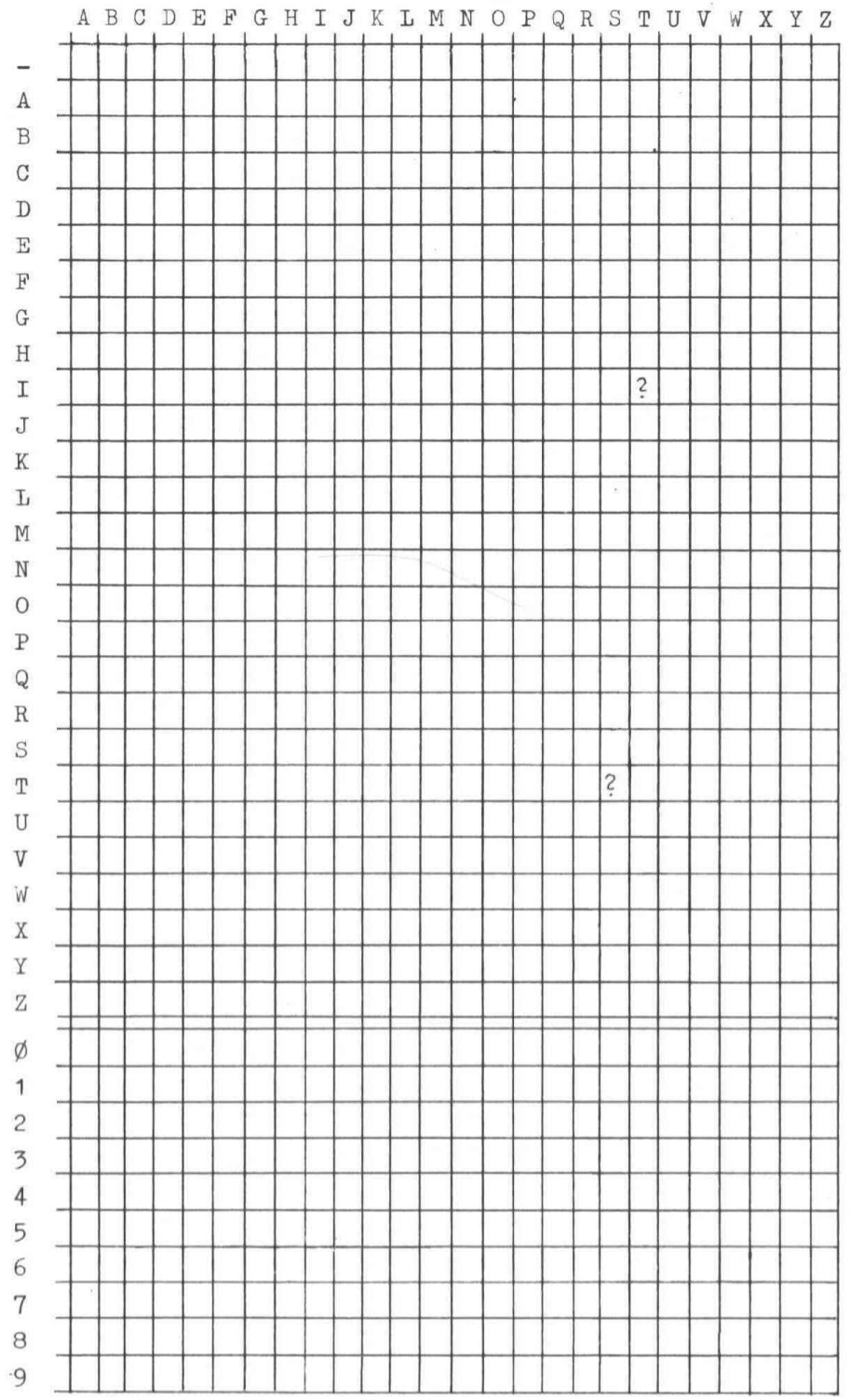


Bild 2.2.1: Variablennamen-Belegungstabelle

Nr.	Gleitkomma				Felder			
	V	FN	%	$	GK	%	$	DIM

Bild 2.2.2: Variablentyp-Tabelle

Nr.	Name	Bedeutung

Bild 2.2.3: Variablenbedeutungs-Tabelle

2.3 BEQUEMES ZEILENLÖSCHEN

Das CLEAR SCREEN oder das fortlaufende WeiterPRINTen (Scrolling) ist oft keine elegante Lösung dafür, daß man bei einem bestehendem Text oder einer Feldmaske auf dem Bildschirm nur eine bestimmte Zeile leeren möchte.

Es gibt zwei VC-20-Maschinensprache-Routinen, die von Basic aus mitbenutzbar sind:

1. Eine ganze Zeile löschen

Nehmen wir an, Sie wollen die 10. Zeile auf dem Bildschirm leeren, dann benutzen Sie im Programm oder im Direkt-Modus:

```
POKE 781,9:SYS 60045
```

In die Zelle #781, die übrigens als X-Register dem Maschinen-Programm ab Adresse #60045 übergeben wird, muß die Zeilennummer eingetragen werden (die Zeilen werden von 0 ab hochgezählt).

2. Eine Zeile teilweise löschen

Nehmen wir weiterhin an, Sie wollen in der 15. Zeile die ersten 5 Zeichenplätze löschen, dann ist folgende Eingabe erforderlich:

```
POKE 781,14:POKE 782,4:SYS 60047
```

Bezgl. Adresse #781 gelten die unter 1. getroffenen Aussagen. In die Zelle #782 wird die Spaltennummer (diese werden ebenfalls von 0 ab hochgezählt), bis wohin gelöscht werden soll, hineingePOKEd, die als Y-Register dem Maschinen-Programm ab Adresse #60047 übergeben wird.

Folgendes kleine Demo-Programm vermittelt Ihnen einen Eindruck von den Anwendungsmöglichkeiten der Routine 60047.

```
10 FORI=1TO505:PRINTCHR$(18)" ";:NEXT
20 FORI=0TO21:POKE781,I+1:POKE782,I:SYS60047:NEXT:WAIT198,1
```

2.4 BEMERKUNGEN ZUM DATASSETTEN-FILEHANDLING

```
0 REM TEST-PGM 1
10 OPEN1,1,1
20 INPUTA,B,C:PRINTA;B;C
30 PRINT#1,A,B,C
40 PRINT#1,A,B,C
50 PRINT#1,A,B,C
60 CLOSE1
70 END
80 :
100 REM TEST-PGM 2A
110 OPEN1
120 INPUT#1,A
130 INPUT#1,B
140 INPUT#1,C
150 PRINTA;B;C
160 CLOSE1
170 END
180 :
200 REM TEST-PGM 2B
210 OPEN1
220 INPUT#1,A,B,C
230 PRINTA;B;C
240 CLOSE1
250 END
300 REM TEST-PGM 3
310 OPEN1
320 INPUT#1,A,B,C
330 INPUT#1,A,B,C
340 INPUT#1,A,B,C
350 PRINTA;B;C
360 CLOSE1
370 END
380 :
400 REM TEST-PGM 4
410 OPEN1
420 GET#1,E$:FL=ST AND 64:IFLEN(E$)=0THEN420
430 PRINTASC(E$);:IFFL=0THEN420
440 CLOSE1
450 END
460 :
500 REM TEST-PGM 5
510 OPEN1,1,1
520 INPUTA,B,C:PRINTA;B;C
530 PRINT#1,A
540 PRINT#1,B
550 PRINT#1,C
560 CLOSE1
570 END
```

Bild 2.4.1

Dem besseren Verständnis des Filehandling (Umgang mit Dateien) in Verbindung mit der Datassette werde dieses Kapitel gewidmet. Gehen wir gleich in medias res und starten das aus mehreren einzelnen Tests bestehende Programm in Bild 2.4.1 mit RUN (Test-PGM 1). Wir werden daraufhin aufgefordert, RECORD & PLAY zu drücken und bald danach 3 Zahlen einzutippen. Als Eingabe wählen wir die Zahlen 1,2,3.

Nach Beendigung des Test-PGMs 1 (Zeile 70) spulen wir das Band zurück zur Ausgangsposition und starten das Test-PGM 2A per RUN 100. Mit Verwunderung werden wir sehen, daß hernach die Zeile 150 die 3 identischen Zahlen 123 "auswirft". Das Test-PGM 2B führt zum gleichen Ergebnis.

Geben wir im Test-PGM 1 die Zahlen -1,-2,-3 ein, so erhalten wir in den Test-PGMen 2A und 2B sogar die Fehlermeldung "FILE DATA ERROR" und im Test-PGM 3 "STRING TOO LONG ERROR".

Die Eingabe -1,2,3 führt in 2A und 2B zur Ausgabe der 3 negativen Zahlen: -123,-123,-123

Was ist geschehen? Des Rätsels Lösung bringt das Test-PGM 4, worin per GET#-Anweisung sämtliche gespeicherten Zeichen des Files einzeln gelesen und deren ASCII-Codes ausgegeben werden. Nach der zuletzt vorgenommenen File-Speicherung mittels PGM 1 wäre die folgende Anzeige das Resultat des PGM 4:

45 49 (13mal:) 32 50 (13mal:) 32 51 32 13
45 49 (13mal:) 32 50 (13mal:) 32 51 32 13
45 49 (13mal:) 32 50 (13mal:) 32 51 32 13

Daraus ist zu erkennen, wie die PRINT#-Zeilen 30-50 des Test-PGMs 1 arbeiten. Die Kommas zwischen A,B,C wirken nämlich nicht als Separatoren in dem Sinne, daß an deren Stelle auf Band CR-Zeichen (#13) gespeichert werden, sondern lediglich mehrere Leerzeichen (#32) entsprechend der Tabulatorfunktion des Kommas bei den PRINT-Anweisungen.

Wird hingegen per RUN 500 im Test-Programm 5 das Abspeichern des Files in der Weise vorgenommen, daß jede einzelne Zahl (z.B. wieder -1,-2,-3) mit einer eigenen PRINT#-Anweisung kombiniert wird, dann liefert Test-PGM 2B das Ergebnis: -1,-2,-3 und Test-PGM 4 die Anzeige der ASCII-Codes:

45 49 32 13 45 50 32 13 45 51 32 13

Die Schlußfolgerung obiger Experimente ist, daß zwar in einer INPUT#-Anweisung mehreren Variablen die von der Datassette kommenden Werte zugewiesen werden können, wie dies Test-PGM 2B beweist (einzelne INPUT#-Anweisungen wie in PGM 2A sind natürlich auch möglich), jedoch nicht in einer PRINT#-Anweisung Werte mehrerer Variablen auf Band aufgezeichnet werden können. Mehrere in einer einzigen PRINT#-Anweisung abgespeicherte Werte werden nämlich nicht durch das CR-Zeichen voneinander getrennt und sind somit als voneinander unabhänige Werte nur durch Tricks wiedererkennbar. Mit anderen Worten müssen einzelne Variablen-Werte durch eigene PRINT#-Anweisungen abgespeichert werden.

2.5 WIE LANGE DAUERT EINE BEFEHLSAUSFÜHRUNG

Normalerweise ist eine Antwort auf diese Frage dann nicht von so großer Bedeutung, wenn keine längeren Verarbeitungszeiten durch Schleifenläufe entstehen. In vielen Fällen - denken wir nur ans Sortieren, Suchen und Berechnen - kommt man jedoch ohne Schleifenläufe nicht aus. Deswegen ist es mitunter recht vorteilhaft zu wissen, wieviel Zeit die eine oder andere Anweisung verbraucht. Diese Kenntnis, gezielt eingesetzt, kann unter Umständen Programmablaufzeiten um 10er-Faktoren verkürzen.

```
0 REM BEFEHLSAUSFUEHRUNGSDAUER
60 TA=TI
70 FOR X=1TO1000:::::::::NEXT
80 T1=TI-TA
100 TA=TI
110 FOR X=1TO1000
120 :
130 :
140 :
200 NEXT
210 PRINT(TI-TA-T1)/60;"MSEC"
```

Bild 2.5.1

```
120 :
130 A=X^2
140 :

120 :
130 A=X*X
140 :

120 :
130 A=X
140 :
```

Bild 2.5.2

Im Programm gemäß Bild 2.5.1 dient Zeile 70 zur Kalibrierung. Es sollten so viele Doppelpunkte untergebracht werden, daß ein RUN dieses Programms nur noch zu einer Ablaufzeit unter 1/10 msec führt. Die Zeilen 120-140 beinhalten die zu messende Anweisung oder auch Kombination von Anweisungen. In Bild 2.5.2 sind Meßbeispiele angegeben.

Die Zuweisung A=X^2 verbraucht 45.3 msec. Dagegen werden bei der Ausführung der mathematisch identischen 2. Zuweisung A=X*X nur 2.9 msec benötigt. Wenn wir zusätzlich berücksichtigen, daß sich beim 3. Test eine Zuweisungszeit von 1 msec ergibt, so erhalten wir für die Ausführung einer Quadrierung ein Netto-Verkürzungsverhältnis von (45.3-1)/(2.9-1)=23.3. Ob eine Berechnung 1 Stunde oder 1 Tag benötigt, dürfte wohl ein großer Unterschied sein.

2.6 STRINGS BEI INPUT UND DATA

Wahrscheinlich haben Sie auch hin und wieder die Fehlermeldungen "EXTRA IGNORED ERROR" oder "REDO FROM START" erhalten, wenn Sie per INPUT oder DATA einer String-Variablen eine Zeichenkette zuweisen wollten. Fehlermeldungen sind vermeidbar, wenn hierbei einige "Spielregeln" beachtet werden, die allerdings weder im VC-20-Begleithandbuch noch im Programmierhandbuch stehen.

Mittels des Einzeiler-Programms:

```
10 INPUT E$,F$:PRINT E$:PRINT F$:RUN
```

lassen sich die folgenden Regeln sofort überprüfen:

1) Strings müssen nur dann in Anführungsstrichen stehen, wenn Kommata oder Doppelpunkte darin enthalten sind. Ein Komma außerhalb von Anführungsstrichen wirkt nämlich als Trennzeichen zwischen Strings, ein Doppelpunkt als Schlußzeichen, so daß alle Zeichen hinter dem Doppelpunkt verworfen werden.

Beispiele:	Eingabe in obigem Programm	Ausgabe
	TEST,WORTE	TEST WORTE
	"TEST1,TEST2:","WORTE"	TEST1,TEST2: WORTE

2) Der letzte einzugebende nicht in Anführungsstrichen stehende String darf nicht durch ein Komma oder Doppelpunkt abgeschlossen werden.

Beispiele:	Eingabe in obigem Programm	Ausgabe
	TEST,WORTE,	?EXTRA IGNORED TEST WORTE

3) Ein String gilt als in Anführungsstrichen stehend nur dann, wenn die Eingabe des Strings durch einen Anführungsstrich eingeleitet wird. Vor dem nachfolgenden Trennzeichen "," (gilt nicht für den als letzten einzugebenden String) muß abermals ein Anführungsstrich stehen. Dieser kann am Ende der letzten oder einzigen Eingabe entfallen. Mit anderen Worten zählen Anführungsstriche als String-Bestandteil, wenn eine String-Eingabe nicht mit einem Anführungsstrich begonnen wird.

Beispiele:	Eingabe in obigem Programm	Ausgabe
	TEST1 "TEST2,"WORTE	TEST1 "TEST2 WORTE

4) Leerzeichen vor dem ersten Zeichen des als ersten einzugebenden Strings zählen nur dann zum String, wenn er in Anführungsstrichen steht. Das gleiche gilt sinngemäß für Leerzeichen hinter dem als letzten einzugebenden String.

Beispiele:	Eingabe in obigem Programm	Ausgabe
	__TEST,__WORTE	TEST WORTE
	"__TEST",WORTE	TEST WORTE

("__" soll bedeuten: Eingabe von zwei Leerzeichen)

Das kleine Programm:

```
10 READ E$,F$:PRINT E$:PRINT F$:DATA (String1),(String2)
```

wird Ihnen bestätigen, daß die obigen Regeln in gleicher Weise für die Stringzuweisung in Verbindung mit DATA und READ gelten.

2.7 TASTATURABFRAGE UND -VERRIEGELUNG

ABFRAGE DER FUNKTIONSTASTEN

In Leserzuschriften ist schon oft danach gefragt worden, wie man in einem Programm die Funktionstasten F1-F8 abfragen kann. Es ist eigentlich recht einfach und verwunderlich, daß im VC-20-Handbuch nichts darüber steht. Eines steht allerdings schon drin, woraus man eigentlich den Abfragealgorithmus ableiten kann. Der ASCII-Tabelle auf den Seiten 145-147 kann nämlich entnommen werden, daß jeder Funktionstasteneingabe ein bestimmter ASCII-Code zugeordnet ist (Bild 2.7.1). Das Programm in Bild 2.7.2 macht sich diese Zuordnung insofern zunutze, als per Funktion ASC der ASCII-Code der

Taste	ASCII-Code (#)	Taste	ASCII-Code (#)
F1	133	F2	137
F3	134	F4	138
F5	135	F6	139
F7	136	F8	140

Bild 2.7.1

```
10 GET E$:IF E$="" THEN 10
20 E=ASC(E$):IF E<133 OR E>140 THEN 10
30 ON 141-E GOSUB 108,106,104,102,107,105,103,101
40 REM PROGRAMM-FORTSETZUNG
50 GOTO 10
101 PRINT "F1":RETURN
102 PRINT "F2":RETURN
103 PRINT "F3":RETURN
104 PRINT "F4":RETURN
105 PRINT "F5":RETURN
106 PRINT "F6":RETURN
107 PRINT "F7":RETURN
108 PRINT "F8":RETURN
```

Bild 2.7.2

Funktionstasteneingabe ermittelt wird und aus diesem Wert der entsprechende ON-GOSUB-Sprungbefehl errechnet wird. Natürlich können die Zeilennummern 101-108 nach Belieben geändert werden, nur ist dabei darauf zu achten, daß die Zuordnung zu den Funktions-Unterprogrammen erhalten bleibt. Auch ist das ON-GOSUB durch ON-GOTO ersetzbar, wenn die RETURNs in GOTO-Sprünge umgewandelt werden.

Unter Umständen benötigen Sie die Funktionstasteneingabe in denjenigen Anwendungsfällen, bei denen das Drücken einer Funktionstaste nur dann eine Wirkung auf den Programmablauf haben soll, wenn Sie sich in einem bestimmten Programmteil "aufhalten". Mit anderen Worten soll das Drücken einer Funktionstaste nicht zwischengespeichert werden. Bei Spielen wird diese Eigenschaft oft verlangt. Da sich die Funktionen GET und INPUT aber auch die im Tastaturpuffer zwischengespeicherten Zeichen holen, wird dies einfach durch die zusätzliche Zeile: 5 POKE 198,0 verhindert.

Diese Methode versagt jedoch dann, wenn der Zustand der gedrückten Taste im Programm berücksichtigt werden soll. Wenn nämlich im oben beschriebenen Algorithmus die Funktionstaste im gedrückten Zustand verbleibt, dann bleibt auch der Tastaturpuffer konstant leer, so daß keine Reaktion im Programm erfolgen kann. Für die Behandlung dieses Falles gibt es noch eine weitere Methode, die im Anhang A.5 nachzulesen ist (Tabelle A.5.9). Mit ihr ist es möglich, jede einzelne Taste auf den Zustand "Gedrückt" oder "Ungedrückt" zu überprüfen. Sie ist zusätzlich mit der Abfrage der Speicherzelle #653 (s. Anhang A.3) koppelbar, so daß jede der 8 Kombinationsmöglichkeiten der gedrückten Tasten SHIFT, CBM und CTRL in Verbindung mit der jeweiligen Funktionstaste unterscheidbar werden. Auf diese Weise gelangen Sie zu insgesamt 8*4=32 anstatt 8 Funktions-"Tasten".

ABFRAGE DER TASTATUR

Im Prinzip können Sie hierbei genauso verfahren wie bei der Abfrage der Funktionstasten, indem Sie das eingegebene Zeichen untersuchen und dann entsprechend im Programm verzweigen.

Das Programm in Bild 2.7.3 verdeutlicht dies. Dort, wo Vergleiche mit darstellbaren Zeichen möglich sind, sollte gemäß Zeile 30

bzw. 40 verfahren werden. Bei Steuertasten wie CURSOR RIGHT oder DELETE kann E$ jedoch nur mit mittels CHR$ erzeugten Strings verglichen werden. Es geht aber auch in der Weise, wie es in den Zeilen 50-60 gezeigt wird.

```
10 GET E$:IF E$="" THEN 10
20 E=ASC(E$)
30 IF E$="1" THEN GOSUB 100
40 IF E$="A" THEN GOSUB 110
50 IF E=148 THEN GOSUB 120
60 IF E=13 THEN GOSUB 130
70 GOTO 10:REM ODER PROGRAMM-FORTSETZUNG
100 PRINT "1":RETURN
110 PRINT "A":RETURN
120 PRINT "DELETE-TASTE":RETURN
130 PRINT "RETURN-TASTE":RETURN
```

Bild 2.7.3

Auch hier könnte es, wie es bei den Funktionstasten schon beschrieben worden ist, vonnöten sein, erst dann eine Eingabe im Programm zu berücksichtigen, wenn die jeweilig geforderte Taste auch wirklich gedrückt wird. Es gelten hierbei die bereits oben getroffenen Ausführungen. Ebenso gilt hier der Verweis auf die Tabelle A.5.9 im Anhang A.5.

TASTATUR VERRIEGELN

Zwecks Verhinderung von Fehleingaben, ungewollten Programmunterbrechungen, Softwareschutz (s. Kap. 6.8) etc. kann es mitunter recht vorteilhaft sein, eine Tastatur-Totalsperre vorzunehmen. Folgendermaßen wird's gemacht:

Verriegelung:	EIN	AUS
	POKE 37154,0	POKE 37154,255

Die Speicherzelle #37154 ist Bestandteil desjenigen VIA-Bausteins, der zur Tastaturabfrage herangezogen wird.

2.8 BEFEHLSEINGABE PER TASTENDRUCK

Daß man sich beim Eintippen eines Programms das Ausschreiben der mitunter recht langen Befehlsworte sparen kann, dürfte hinreichend bekannt sein. Es gibt ja die Anweisungs-Kürzel auf S. 133 im VC-20-Handbuch. Am meisten hat sich da wohl das "?" durchgesetzt, welches die Eingabe der Anweisung PRINT ersetzt.

Noch eleganter wäre die Eingabe von Anweisungen jedoch, wenn dies auf Tastendruck geschehen könnte. Die Konkurrenten ZX 81 sowie ZX Spectrum können das ja auch, warum also nicht der VC-20? Doch, er kann! Mit dem Programm BEFEHL-TASTEN in Bild 2.8.1 rufen Sie die gebräuchlichsten Befehls- und Funktionsworte auf per:

SHIFT + Buchstabe

```
10 REM BEFEHL-TASTEN
20 REM SHIFT+BUCHSTABE=BEFEHL
30 L=89
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANFANGSADRESSE AENDERN!":STOP
230 IFCH<>11873THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 SYSSD:NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(146):PRINT:RETURN
1000 DATA162,L12,160,H,120,142,20,3,140,21,3,96,165,215,197,163
1010 DATA208,3,76,191,234,133,163,201,193,144,247,201,219,176,243,56
1020 DATA233,193,170,189,L63,H,168,162,0,134,198,169,20,72,41,127
1030 DATA230,198,157,119,2,104,48,218,232,185,158,192,200,208,238
2000 DATA231,128,234,96,238,3,133,65,20,35,51,111,249,6,124,92
2010 DATA217,243,224,151,56,228,10,28,44,121
```

Bild 2.8.1

Das Maschinenprogramm in BEFEHL-TASTEN wird je nach Wunsch in einem beliebigen Speicherbereich generiert. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen Überschreiben geschützt und sie können es auch dann aufrufen, wenn Sie beliebige andere Programme im Arbeitsspeicher haben. Weiterhin ist es koppelbar mit von Ihnen frei definierbaren Befehlen (s. Kap. 4.1).

BEDIENUNGSHINWEISE

BEFEHL-TASTEN enthält ein Maschinen-Programm und ist ein Basic-Loader, der vom Prinzip her der gleiche wie der im Kapitel 5.2 beschriebene ist. Detailinformation bezgl. diesem kann dort in Erfahrung gebracht werden. Der Basic-Loader wird normal mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das M-PGM generiert werden soll. Bei Eingabe von "E" wird es ans Basic-RAM-Ende gespeichert, wobei eventuell schon andere vorhandene M-PGMe in keinster Weise angetastet werden.

Nachdem Sie sich zum Beispiel für letztere Eingabe-Möglichkeit entschieden haben, wurden Sie bei einem VC-20 in GV die Anzeige erhalten:

```
M-PGM-AUFRUF MIT:
SYS 7584
```

Die SYS-Adresse schreiben Sie sich am besten irgendwo auf. Die "Befehl-Tasten" sind jetzt aktiviert und Sie können per SHIFT + Buchstabe den aus Bild 2.8.2 abzulesenden Befehl auf den Bildschirm zaubern. Sie werden schnell Routine mit den am Anfang ungewohnten Zuordnungen entwickelt haben. Außerdem stimmen in gut der Hälfte der Falle die Buchstaben mit den Anfangsbuchstaben der Anweisungen überein.

A	ASC	F	FOR	K	GOSUB	P	POKE	U	RETURN
B	CLOSE	G	GET	L	LIST	Q	PEEK	V	VAL
C	CHR$	H	STOP	M	MID$	R	RIGHT$	W	DATA
D	PRINT#	I	INPUT	N	NEXT	S	STR$	X	READ
E	LEFT$	J	GOTO	O	OPEN	T	THEN	Y	RESTORE
								Z	SYS

Bild 2.8.2

```
2000 LDX #$0C          IRQ-Vektor auf $200C setzen
2002 LDY #$20
2004 SEI
2005 STX $0314
2008 STY $0315
200B RTS
-------------------------------------------------------------
200C LDA $D7           Letztes ausgegebenes Zeichen
200E CMP $A3           Vergleich mit gerettetem Zeichen
2010 BNE $2015         Sprung, wenn Zeichen wechselt
2012 JMP $EABF         Interruptroutinen-Umleitungsende
2015 STA $A3           Ausgegebenes Zeichen retten
2017 CMP #$C1
2019 BCC $2012         "Filter" fuer gesHIFTete Buchstaben
201B CMP #$DB
201D BCS $2012
201F SEC
2020 SBC #$C1
2022 TAX               #$C1 abziehen => A=00,B=01,...
2023 LDA $203F,X       Offset fuer Befehlstabelle
2026 TAY
2027 LDX #$00          Tastaturpuffer initialisieren
2029 STX $C6
202B LDA #$14          DELETE = 1.Zeichen im Tastaturpuffer
202D PHA               Zeichen retten
202E AND #$7F          Bit 7, falls vorhanden, entfernen
2030 INC $C6
2032 STA $0277,X       Zeichen im Tastaturpuffer ablegen
2035 PLA               gerettetes Zeichen holen
2036 BMI $2012         fertig, wenn Bit 7 gesetzt
2038 INX               Index X erhoenen
2039 LDA $C09E,Y       Befehlszeichen holen
203C INY               Index Y erhoehen
203D BNE $202D         Sprung: immer
-------------------------------------------------------------
203F E7 80 EA 60 EE    ASC     CLOSE   CHR$    PRINT# LEFT$
2044 03 85 41 14 23    FOR     GET     STOP    INPUT  GOTO
2049 33 6F F9 06 7C    GOSUB   LIST    MID$    NEXT   OPEN
204E 5C D9 F3 E0 97    POKE    PEEK    RIGHT$  STR$   THEN
2053 38 E4 0A 1C 2C    RETURN  VAL     DATA    READ   RESTORE
2058 79                SYS
```

Bild 2.8.3

PROGRAMM-EINZELHEITEN

1.) Name: BEFEHL-TASTEN

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Basic-Loader

4.) Anzahl der Bytes
    des Basic-Loaders: 1168
    des M-PGMs: 89

5.) Benutzte Variablen: s. Kap. 5.2

6.) Listings
    des Basic-Loaders: Bild 2.8.1
    des M-PGMs: Bild 2.8.3

7.) Erläuterungen zum Basic-Loader werden in Kap. 5.2 gegeben und zum M-PGM implizit im Listing (Bild 2.8.3).

3

Abspeichern und Laden

3.1 DER KASSETTENPUFFER

Der Kassettenpuffer ist ein 192 Byte großer vom Betriebssystem vor dem Überschreiben geschützter RAM-Bereich, der für alle Kassetten-Operationen benutzt wird. In der Einschaltroutine (RESET) des VC-20 wird die Anfangsadresse des Kassettenpuffers auf $033C = #828 festgelegt und in die Zero-Page-Adressen $B2,B3 = #178,179 geschrieben. Durch Ändern der Adressen $B2,B3 kann die Position des Kassettenpuffers innerhalb des RAM-Bereichs verschoben werden.

Als Puffer, also Zwischenspeicher für Daten, wird dieser Bereich allerdings nur bei PRINT#-, GET#- und INPUT#-Anweisungen benutzt, sofern sich diese auf die Band-Kassette beziehen. In allen anderen Fällen (SAVE, LOAD, OPEN) enhält der Kassettenpuffer den Tape-Header (s. Kap. 3.2) - dem fast schon zwanghaft erscheinendem Übersetzungstrieb schließen wir uns lieber nicht an, denn mit einem Versuch wie "Band-Kopfdaten" in diesem Fall wird Ihnen, verehrte Leser, nicht besonders gedient sein, oder nur insofern, als wir Ihnen ein erfrischendes Schmunzeln entlocken könnten.

In der Funktion als Puffer wird bei einer PRINT#-Anweisung das auszugebende Zeichen im Kassettenpuffer zwischengespeichert. Erst wenn dieser mit 192 Zeichen gefüllt ist oder die CLOSE-Anweisung gegeben wird, wird ein Daten-Block auf die Kassette geSAVEd. Im umgekehrten Fall, also bei einer INPUT#- oder GET#-Anweisung werden 192 Zeichen in den Kassettenpuffer geladen und von dort aus einer Variablen zur Weiterverarbeitung übergeben. Erst wenn alle Zeichen ausgelesen sind, wird der nächste Datenblock geladen.

Der Kassettenpuffer ist außerdem ein Bereich, in dem sehr gerne kleine Maschinensprache-Routinen abgelegt werden (s. Kap. 5.2). Diese können zusammen mit dem File-Namen in den Kassettenpuffer geladen, oder im B-PGM mithilfe eines Basic-Loaders dorthin gePOKEd werden (s. Kap. 6.8 "Softwareschutz").

3.2 TAPE-HEADER

Der Tape-Header steht als Vorspann vor jedem PGM oder Daten-File auf der Kassette. Er ist 192 Bytes lang und enthalt alle Angaben über das folgende PGM.

Diese Angaben sind:

1. Das File-Kennzeichen

   Dieses Kennzeichen für den Typ des folgenden PGMs steht im 1. Byte des Tape-Headers und ist abhängig von der Sekundär-Adresse im SAVE-Befehl. Die Bedeutung des File-Kennzeichen in Abhängigkeit von der Sekundär-Adresse können Sie dem Bild 3.2.1 entnehmen.

Sek.-Adr.	File-Kennz.	Bedeutung
0 (ohne)	1	Basic-PGM
1	3	Maschinenspr.-PGM
2	1	Basic-PGM mit EOT
3	3	Maschinenspr.-PGM mit EOT

EOT = Bandende-Kennzeichen (End of Tape)

Bild 3.2.1: File-Kennzeichen

2. Die Länge des PGMs:

   Die Bytes 2-5 des Tape-Headers enthalten die Adressen, zwischen denen das PGM stand, als es geSAVEd wurde. Die Anfangsadresse steht in den Bytes 2 und 3, die Endadresse in den Bytes 4 und 5 (in der Reihenfolge: L,H).

3. Der PGM-Name:

   Die Bytes 6-192 des Tape-Headers enthalten den File-Namen. Er darf bis zu 187 Zeichen lang sein, wovon bei LOAD aber nur 16 Zeichen dargestellt werden.

Der LOAD-Befehl liest den Tape-Header in den Kassettenpuffer und vergleicht den File-Namen mit dem Namen im LOAD-Befehl. Anschließend wird, abhangig vom File-Kennzeichen, das PGM in den Arbeitsspeicher geladen.

Anhand folgender Beispiele soll das Zustandekommen des File-Kennzeichens und dessen Bedeutung etwas naher beschrieben werden.

BEISPIELE:

SAVE "TEST",1,0 Ein Basic-PGM wir unter dem Namen "TEST" auf Kassette geSAVEd, das File-Kennzeichen ist 1. Da beide Werte, die Gerätenummer 1 sowie die Sekundar-Adresse 0, Default-Werte sind, können sie in diesem Fall auch weggelassen werden.

SAVE "TEST1",1,1 Ein PGM, dessen Anfangs- und Endadresse in den Zero-Page-Adressen $2B-2E stehen, wird unter dem Namen "Test1" auf Kassette geSAVEd, das File-Kennzeichen ist 3.
Ein auf diese Weise geSAVEtes PGM wird mit dem LOAD-Befehl immer an die Adresse geladen, die im TAPE-Header steht (absolutes Laden), d.h.:
PGMe die ab Adresse $1200 (#4608) standen, werden immer nach Adresse $1200 geladen, egal, ob der VC-20 in der Grundversion oder mit einer Erweiterung (3K oder 8K) betrieben wird, oder wie sonst die Vektoren $2B,2C und $2D,2E stehen.

SAVE "TEST2",1,2 Ein B-PGM wird unter dem Namen "Test2" auf Kassette geSAVEd, das File-Kennzeichen ist 1. Nach dem PGM wird ein Bandende-Kennzeichen geSAVEd. Die Hinweise bezüglich des Bandende-Kennzeichen im VC-20-Handbuch sind falsch.

Wenn ein PGM oder eine Datei geladen werden soll und der VC-20 ein Bandende-Kennzeichen liest, so wird der Suchvorgang abgebrochen und die Fehlermeldung "DEVICE NOT PRESENT" ausgegeben (leider nicht "FILE NOT FOUND").

Der Befehl LOAD "TEST" veranlaßt den VC-20, nach einem PGM mit dem Namen "TEST" zu suchen und dieses zu laden. Es wird aber jedes PGM geladen, dessen Name mit "TEST" beginnt, also auch "TEST1", "TESTPGM" usw. Man muß das beachten, wenn eine Kassette nach einem bestimmten PGM oder einer Datei durchsucht werden soll.

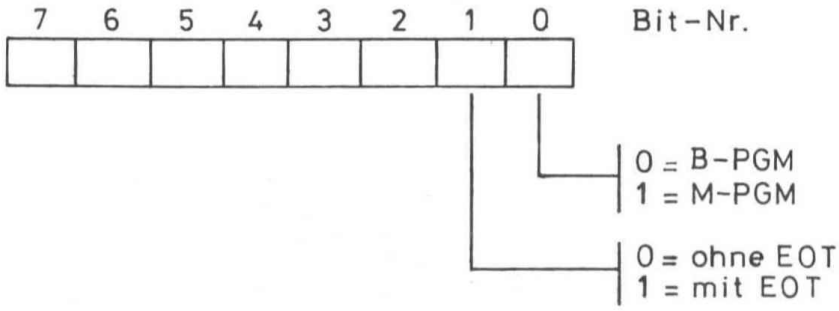


Bild 3.2.2: Sekundaradresse

Alles, was hier uber den SAVE- bzw. LOAD-Befehl geschrieben wurde, gilt sinngemaß auch für die OPEN-Anweisung.
Ausnahmen:

Sekundär-Adresse	Bedeutung
0	Datei lesen
1	Datei schreiben
2	Datei schreiben und nach der CLOSE-Anweisung ein Bandende-Kennzeichen speichern

Das File-Kennzeichen für eine Datei ist 4. Immer wenn nach dem LOAD-Befehl ein Tape-Header gefunden wird, dessen File-Kennzeichen eine 4 ist, wird "FOUND (Name)" ausgegeben, aber das File nicht geladen.

Bild 3.2.2 zeigt noch einmal die bitweise Auswertung der Sekundär-Adresse bei SAVE-Befehlen.

Übrigens, LOAD "",1,1 lädt ein PGM an diejenige Adresse, die im Tape-Header steht, egal, ob das File-Kennzeichen 1 oder 3 ist (absolutes Laden). Diese Tatsache kommt also dem Fall gleich, daß ein PGM mit SAVE "(Name)",1,1 abgespeichert worden ist und einfach mit LOAD (also ohne zusätzliche Adressen) geladen wird.

Bei Floppys wäre zusätzlich zu beachten, daß ein einfaches LOAD "(Name)",8 nicht ausreicht, um ein PGM absolut zu laden. Auch in diesem Fall ist die Sekundäradresse 1 zu verwenden: LOAD "(Name)",8,1.

Das folgende PGM ist nach den Beschreibungen dieses und des Kapitels 3.1 "Der Kassettenpuffer" geschrieben worden. Ein PGM-Beschreibung ist daher überflüssig.

Laden Sie das PGM (s. Bild 3.2.3), legen Sie eine beliebige Kassette in Ihre Datassette und starten Sie das PGM mit RUN. Als Ergebnis erhalten Sie auf dem Bildschirm ein Inhaltsverzeichnis Ihrer Kassette, das neben dem PGM-Namen auch den PGM-Typ, dessen Länge und bei M-PGMen auch deren Anfangsadressen enthält.

```
0 REM TAPE-HEADER
10 PRINT"■":REM CLR-HOME
20 Z=Z+1:REM PGM-NUMMER
30 OPEN1,1,0:REM TAPE-HEADER LESEN
40 T=PEEK(828):REM FILE-TYP
50 IFT=1THENT$="BASIC-PGM"
60 IFT=3THENT$="MASCH.-PGM"
70 IFT=4THENT$="DATEN-FILE"
80 A=PEEK(829)+PEEK(830)*256:REM ANF.ADRESSE
90 E=PEEK(831)+PEEK(832)*256:REM ENDADRESSE
100 L=E-A:REM PGM-LAENGE
110 N$="":FORI=833TO1019
120 N$=N$+CHR$(PEEK(I)):REM FILE-NAME
130 NEXT
140 PRINT"■■":REM 2*CRSR DWN
150 PRINTZ".PROGRAMM"
160 PRINT"TYP      : ";T$
170 IFT<>4THENPRINT"LAENGE   :";L;"BYTE"
180 IFT=3THENPRINT"ANF.ADR.:";A
190 PRINT"NAME     : ";N$
200 CLOSE1:GOTO20
```

Bild 3.2.3

3.3 LADEN/SPEICHERN NICHT IMMER MIT LOAD/SAVE

In den VC-20-Kreisen hat es sich schon längst herumgesprochen: mit LOAD und SAVE kommt man desöfteren beim Laden und Abspeichern nicht aus. Vor allem gibt es dabei mit der Datassette Ärger, weswegen im weiteren nur auf die Probleme bezüglich dieser eingegangen wird. Das hier Gesagte kann jedoch mit einigen Zusätzen (z.B. Geratenummer 8, etc.) ebenso auf die VC-20-Floppy-Bedienung übertragen werden.

In vielen Fällen tritt ein SAVE-Problem bei denjenigen GV-Programmen auf, die den Arbeitsspeicher bis zum Speicherplatz-"Rand" beanspruchen. Das sind diejenigen Programme, die genau 3583 Bytes lang sind und im Bereich $1001-1DFF bzw. #4097-7679 liegen. Zwar ist es hierbei moglich, sie einfach mit SAVE ohne Namensgebung abzuspeichern. Was macht man jedoch, wenn diese **mit** Namen geSAVEd werden sollen. Folgender Trick hilft in diesen Fällen:

- Leeren Sie den Bildschirm mit SHIFT + CLEAR.

- Fahren Sie mit dem Cursor zur Mitte des Bildschirms (auf eine Zeile mehr oder weniger kommt es hierbei nicht an).

- Speichern Sie das Programm mit folgender Zeile ab:

```
POKE 55,55:SAVE "(Name)"     (Return)
```

Mit POKE 55,55 blähen Sie namlich Ihren GV-Arbeitsspeicherbereich um 55 weitere Bytes auf, die ihrerseits jedoch im oberen Bildschirmbereich liegen. Damit Ihre Eingabe nicht mit diesen Speicherzellen ins "Gerangel" kommt, muß sie weiter unten im Bildschirm vorgenommen werden.

Dies war ein Beispiel für ein SAVE-Problem. Ein typisches Ladeproblem ist folgendes: nehmen wir an ein Programm "läuft" nur innerhalb eines bestimmten Speicherbereichs Ihres Rechners - die im Programm verwendete Zeilennummer hat nichts mit der Position des Programms im Speicher zu tun! -, dann wird diesem Fall durch die anfänglich vorgegebene Festlegung des Arbeitsspeichers seitens des VC-20-Betriebssystems (s. Kap. 2.1) wohl kaum Rechnung getragen. Es ist somit notwendig, den Arbeitsspeicher zu versetzen. Im

Kapitel 2.1 wird dies eingehend erläutert. Hier nur ein Beispiel: wenn das Programm ab $2001=#8193 laufen soll und sie es dorthin laden wollen, dann geht das so:

POKE 44,32:POKE 8192,0:NEW und danach: LOAD

Schwierigkeiten entstehen oft auch beim Laden und Abspeichern von Maschinenprogrammen, insbesondere von Modulprogrammen (s. Kap. 6.5). Das folgende Verfahren hat sich fur die meisten Falle als annehmbar und vor allen Dingen als funktionsfähig erwiesen:

1. Maschinenprogramme abspeichern

Zu allererst muß der genaue "Standort" des Programms und dessen Länge bekannt sein. Gehen wir von folgendem Beispiel aus: wir nehmen an, daß sich im Bereich $A000-BFFF ein Spielprogramm befindet, das somit eine Länge von 8192 Bytes hat. Mit der Dassette gibt es keine Möglichkeit (jedenfalls haben wir bis heute keine erfahren) ein solches Programm direkt zu SAVEn, weil das VC-20-Betriebssystem hierbei einen "Riegel" vorschiebt (komischerweise ist das bei der Floppy nicht so). Das Programm muß also erst in einen SAVEbaren Speicherbereich übertragen werden. Sie können zum Beispiel das Programm VC-TRANSFER/COMPARE/FILL aus Kapitel 5.4 zu diesem Zweck einsetzen. Im weiteren gehen wir davon aus, daß es nach Übertragung im Bereich $1E00-3DFF bzw. #7680-15871 liegt. Da wir von Kapitel 2.1 her wissen, daß beim SAVEn alles zwischen dem Basic-Anfang (Vektor #43,44) und dem Variablen-Anfang (Vektor #45,46) abgespeichert wird, ist die folgende Eingabe wohl die logische Konsequenz:

```
POKE 43,0:POKE 44,30:POKE 45,0:POKE 46,62:CLR:SAVE "(Name),1,1"
```

Die Erklärung für die letzten Zeichen ",1,1" entnehmen Sie bitte dem vorigen Kapitel 3.2. So viel sei aber hier gesagt. Es wird dadurch das File-Kennzeichen 3 auf Band gespeichert, daß vor allem beim Laden des Programms wichtig ist. Das Betriebssystem "weiß" dann nämlich, daß kein Basic-Programm vorliegt und rechnet deswegen auch nicht die Links (s. Kap. 2.1) um.

Wie Sie in der letztgenannten Eingabe unschwer erkennen können, handelt es sich bei den POKE-Werten von #43,44 sowie #45,46 um

die LOW/HIGH-Anteile der Anfangsadresse #7680 und der Endadresse (+1 !!) #15872. Das CLR ist fur die entsprechende Anpassung der restlichen Vektoren erforderlich.

2. Maschinenprogramme laden

Dies ist mit weniger Mühen verbunden: wenn es sich um Maschinenprogramme handelt, die ursprünglich innerhalb des Bereiches bis $7FFF lagen, so reicht das LOAD aus. Denn das File-Kennzeichen 3 sorgt nicht nur dafur, daß Veränderungen bei den zu ladenden Bytes (Link-Umrechnung) verhindert werden, sondern auch daß die eingelesenen Bytes dorthin gespeichert werden, von wo sie "hergeholt" (SAVE-Prozedur) worden sind.

Die letzte Aussage heißt mit anderen Worten aber auch, daß das im oben beschriebenen Beispiel abgespeicherte Spielprogramm wieder in den Bereich #7680-15871 zurückgeladen wird. Das ist weiter nicht schlimm, weil wir in umgekehrter Weise von VC-TRANSFER Gebrauch machen und das Spielprogramm nach $A000-BFFF kopieren können. Es gibt da aber ein einfacheres Verfahren:
Nachdem die Kassette mit dem Spielprogramm richtig positioniert worden ist, geben Sie ein:

SYS 63407 (Return)

Der Aufforderung PRESS PLAY ON TAPE kommen Sie nach. Diese Eingabe sorgt jedoch nur dafür, daß der Tape-Header (s. Kap. 3.2) des Spielprogramms geladen wird. Dadurch, daß Sie zu diesem Zeitpunkt die für den Ladeprozeß maßgebliche Anfangsadresse (LOW/HIGH-Format in #829,830) sowie Endadresse (+2 !!) (LOW/HIGH-Format in #831,832) "verstellen" können, ist es Ihnen somit möglich, sich den gewünschten Zielspeicherbereich frei auszusuchen. Wir wählen natürlich $A000-BFFF und geben deswegen ein:

POKE 830,160:POKE 831,1:POKE 832,192:SYS 62980 (Return)

Das SYS 62980 setzt den Ladeprozeß fort, nach dessen Beendigung Sie noch NEW eingeben. Im Falle eines Autostart-Modulprogramms starten Sie es mit SYS 64802 oder RESET.

4

Zusätzliche Basic-Funktionen

4.1 EIGENE BEFEHLE DEFINIEREN

Fast jeder von Ihnen hat schon mal das Stichwort "Extended Basic" oder "Exbasic" irgendwo gelesen oder gehört. Was damit gemeint ist, sagen diese Begriffe implizit aus: es handelt sich um Basic, welches um mehrere Befehle erweitert worden ist. Per Kassette, Diskette oder Steckmodul kann man Basic-Erweiterungen inzwischen fast überall käuflich erwerben. Für viele Programmierer wäre es aber viel interessanter, diesbezüglich nicht auf fertige Produkte zurückgreifen zu müssen, sondern vielmehr die Möglichkeit zu haben, sein eigenes individuell ausgeprägtes Exbasic-Profil zu kreieren.

Das in diesem Kapitel beschriebene Programm VC-COMMAND macht's möglich. Mit dessen Hilfe können Sie jedes beliebige Maschinenprogramm, welches Sie normalerweise per SYS-Befehl aufrufen, einem von Ihnen gewählten Befehlswort zuordnen. Bei Eingabe dieser selbst definierten Befehle im Direkt-Modus oder auch bei deren Benutzung im Programm wird der diesem Befehl zugeordnete SYS-Befehl von selbst ausgeführt. Der Benutzer braucht sich demzufolge nicht mehr die Adresse zu merken, auf die er sonst per SYS springen müßte, sondern nur noch die selbst gewählten einfach einprägsamen Befehlsworte.

Das Maschinenprogramm VC-COMMAND (s. Bild 4.1.3) wird je nach Ihrem Wunsch in einem beliebigen Speicherbereich generiert. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen Überschreiben geschützt.

BEDIENUNGSHINWEISE

VC-COMMAND ist ein Maschinen-Programm, das in den im Kapitel 5.2 beschriebenen Basic-Loader (Bild 4.1.1) eingebunden wurde. Detailinformation bezgl. diesem kann dort in Erfahrung gebracht werden. Für VC-COMMAND sind einige Programmzeilen hinzugekommen, die unter 7.) bei den Programm-Einzelheiten kommentiert werden. Bevor Sie den Basic-Loader normal mit RUN starten, sollten Sie ihn abgeSAVEd haben, weil dieser sich nach getaner Arbeit von selbst in Wohlgefallen auflöst. Zu Anfang werden Sie gefragt, ab welcher Adresse (dezimal) das M-PGM generiert werden soll. Bei

Eingabe von "E" wird es ans Basic-RAM-Ende gespeichert, wobei eventuell schon andere vorhandene M-PGMe in keinster Weise angetastet werden.

```
10 REM VC-COMMAND
15 REM EINBAU VON ZUSAETZLICHEN BEFEHLEN
20 REM IN DATA'S AB 1000: (BEFEHLSNAME),(DEZ-EINSPRUNGADRESSE)
25 REM LETZTER NAME MUSS "@" SEIN
27 REM BEFEHL "OFF" SETZT BASIC-VEKTOREN ZURUECK
30 L=122
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 GOSUB500:INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L-TL:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>15555THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 GOSUB600:SYSSD:NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(146):PRINT
280 PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
500 READCM$:LE=LEN(CM$):IFCM$="@"THENRETURN
510 TL=TL+LE+3:READES:GOTO500
600 RESTORE:TB=I-1
610 READCM$:LE=LEN(CM$):IFCM$="@"THENPOKETB,0:RETURN
620 FORI=0TOLE-1:POKETB+I,ASC(MID$(CM$,I+1,1)):NEXT
630 POKETB+LE,0:READAD:AD=AD-1:GOSUB250
640 POKETB+LE+1,AH%:POKETB+LE+2,AL
650 TB=TB+LE+3:GOTO610
1000 DATA @
2000 DATA162,L11,160,H,142,8,3,140,9,3,96,32,115,0,32,L20
2010 DATAH,76,174,199,240,244,233,128,144,3,76,243,199,162,L115,160
2020 DATAH,134,187,132,188,160,0,152,24,101,187,133,187,169,0,168
2030 DATA101,188,133,188,177,187,240,51,209,122,208,35,200,177,187,240
2040 DATA7,209,122,208,26,200,208,245,152,24,101,122,133,122,169,0
2050 DATA101,123,133,123,200,177,187,72,200,177,187,72,76,121,0,177
2060 DATA187,240,3,200,208,249,200,200,200,208,188,104,104,32,121,0
2070 DATA76,231,199,79,70,70,0,228,90,0
```

Bild 4.1.1

Nachdem Sie die gewünschte Anfangsadresse bzw. "E" eingegeben und mit RETURN bestätigt haben, wird das M-PGM VC-COMMAND abgespeichert, und die von Ihnen vorgegebenen Befehlsworte werden aktiviert. Bevor der Basic-Loader sich selbst löscht, wird auf dem Bildschirm derjenige SYS-Befehl angezeigt, der die Befehle von neuem aktiviert, wenn diese zwischendurch deaktiviert wurden. Diesen Befehl schreiben Sie sich zweckmäßigerweise auf.

Die Zuordnungen von Befehlsworten und SYS-Adressen müssen als DATA-Zeilen vor Zeile 2000 in den Basic-Loader (Bild 4.1.1) eingefügt werden. In den DATA-Zeilen stehen hintereinander die Paare von Befehlswort und SYS-Adresse. Die Liste muß mit dem "Klammeraffen"-Zeichen als Endekennzeichen abgeschlossen werden. Ein Beispiel für eine solche DATA-Liste liefert Bild 4.1.2.

```
1000 DATA FERTIG,51249:REM END
1010 DATA AB,51010:REM FOR
1020 DATA WEITER,52510:REM NEXT
1040 DATA EINGABE#,52133:REM INPUT#
1050 DATA EINGABE,52159:REM INPUT
1060 DATA LIES,52230:REM READ
1070 DATA LASS,51621:REM LET
1080 DATA ZU,51360:REM GOTO
1090 DATA LOS,51313:REM RUN
1100 DATA WENN,51496:REM IF
1110 DATA DATENANFANG,51229:REM RESTORE
1120 DATA UP,51331:REM GOSUB
1130 DATA RAUS,51410:REM RETURN
1140 DATA ',51515:REM REM
1150 DATA ←,51247:REM STOP
1160 DATA BEI,51531:REM ON
1170 DATA WARTE,55341:REM WARTE
1180 DATA LADE,57701:REM LOAD
1190 DATA RETTE,57683:REM SAVE
1200 DATA PRUEFE,57698:REM VERIFY
1210 DATA IN,55332:REM POKE
1220 DATA AUS#,51840:REM PRINT#
1230 DATA AUS,51872:REM PRINT
1240 DATA PGM,50844:REM LIST
1250 DATA LOESCH,50782:REM CLR
1260 DATA MP,57639:REM SYS
1270 DATA DATEI,57787:REM OPEN
1280 DATA SCHLIESS,57796:REM CLOSE
1290 DATA HOLE,52091:REM GET
1300 DATA NEU,50754:REM NEW
1310 DATA @
```

Bild 4.1.2

In diesem Beispiel (Bild 4.1.2) wurde, was unschwer zu erkennen ist, die Absicht verfolgt, die im VC-20 eingebauten Basic-Befehle zu verdeutschen bzw. zu symbolisieren (z.B.: ' für REM). Werden die DATA-Zeilen dieses Beispiels in den Basic-Loader eingebunden, so sind nach Aktivierung neben den bisher gewohnten Befehlen zusätzlich die neu hinzugekommenen benutzbar. Anstatt RUN kann also auch LOS eingegeben werden.

Ein Befehl ist von vornherein und unabhängig von jeglicher Befehlsdefinition gültig. Das ist der Befehl OFF. Dieser bewirkt, daß der Basic-Interpreter-Vektor auf seinen Normalwert zurückgesetzt wird, so daß die zusätzlichen Befehle dem VC-20 nicht mehr verständlich sind. Testen Sie OFF, indem Sie den Basic-Loader gemäß Bild 4.1.1 starten und hinterher OFF eingeben. Es wird die READY-Meldung folgen. Geben Sie dann erneut OFF ein, so ist dieser Befehl natürlich nicht mehr aktiv und das Resultat ist SYNTAX ERROR.

Bei der Definierung von neuen Befehlen ist zu beachten, daß syntaktische Probleme auftreten können. VC-COMMAND funktioniert nämlich in der Weise, daß ein vorkommender Befehl - ob im Direkt-Modus oder im Programm - zuerst daraufhin überprüft wird, ob dieser zu einen der "eingebauten" gehört. Ist dies nicht der Fall, dann wird der Befehl mit den neu hinzugekommenen verglichen. Wenn allerdings auch dann keine Identifizierung moglich ist, so erscheint die Meldung SYNTAX ERROR. Die syntaktischen Probleme tauchen dann auf, wenn die Zeichenkette eines der "eingebauten" Befehle in der Zeichenkette eines neuen Befehls enthalten ist. Beispielsweise würde der neue Befehl FORGET - was auch immer er für eine Funktion hätte - nicht als neuer Befehl identifizierbar sein. Noch schlimmer: er würde sogar immer die Fehlermeldung SYNTAX ERROR verursachen, weil der VC-20 nach dem Befehl FOR nicht einen weiteren Befehl (GET), sondern eine Variable erwartet.

Weiterhin ist bei der Befehlsliste in den DATA-Anweisungen zu berücksichtigen, daß VC-COMMAND die vorgegebenen Befehle mit dem eingegebenen in der Reihenfolge vergleicht, wie sie in den DATA-Zeilen stehen. Was also zuerst erkannt worden ist, wird auch zuert abgearbeitet. Aus diesem Grund dürften die DATA-Zeilen 1220 und 1230 nicht ausgetauscht werden, weil in diesem Fall bereits beim Befehl AUS bei Eingabe von AUS# die Suche erfolgreich wäre.

```
2000 LDX #$0B          BASIC-Interpreter umleiten
2002 LDY #$20
2004 STX $0308
2007 STY $0309
200A RTS
200B JSR $0073         Naechstes Zeichen holen
200E JSR $2014         Syntax Pruefen
2011 JMP $C7AE         zum Anfang der Interpreterroutine
2014 BEQ $200A         Sprung, wenn ":" oder Zeilenende
2016 SBC #$80
2018 BCC $201D         Sprung, wenn Zuweisung oder Zusatzbefehl
201A JMP $C7F3         Sprung, wenn 1. Zeichen = BASIC-Token
201D LDX #$73          Tabellenvektor $BB,$BC initialisieren
201F LDY #$20
2021 STX $BB
2023 STY $BC
2025 LDY #$00          Y=0
2027 TYA               Tabellenvektor um Y vergroessern
2028 CLC
2029 ADC $BB
202B STA $BB
202D LDA #$00
202F TAY               Y=0
2030 ADC $BC
2032 STA $BC
2034 LDA ($BB),Y       1. Zusatzbefehlszeichen aus Tab. holen
2036 BEQ $206B         Sprung, wenn 0
2038 CMP ($7A),Y       mit 1. Eingabezeichen vergleichen
203A BNE $205F         Sprung, wenn ungleich
203C INY               Y=Y+1
203D LDA ($BB),Y       Naechstes Zusatzbefehlszeichen holen
203F BEQ $2048         Sprung, wenn Vergleich erfolgreich
2041 CMP ($7A),Y       mit naechstem Eingabezeichen vergleichen
2043 BNE $205F         Sprung, wenn ungleich
2045 INY
2046 BNE $203D         Sprung: immer
2048 TYA               Y zu Zeilennummer addieren
2049 CLC
204A ADC $7A
204C STA $7A
204E LDA #$00
2050 ADC $7B
2052 STA $7B
2054 INY               Sprungadresse aus Tabelle holen und
2055 LDA ($BB),Y           in Stack legen
2057 PHA
2058 INY
2059 LDA ($BB),Y
205B PHA
205C JMP $0079         Sprung nach vorgegebener Sprungadresse
```

Bild 4.1.3 (Teil 1)

```
205F LDA ($BB),Y      Zusatzbefehlsende suchen
2061 BEQ $2066        Sprung, wenn gefunden
2063 INY
2064 BNE $205F
2066 INY              Sprungadresse ueberlesen
2067 INY
2068 INY
2069 BNE $2027        Sprung: immer
206B PLA              RuecksprungadresSe loeschen
206C PLA
206D JSR $0079        letztes Zeichen holen
2070 JMP $C7E7        EinsPrung in VC20-InterPreterroutine
```

```
2073 4F 46 46 00 E4    "OFF" (0=Befehlsende)   HIGH-
2078 5A 00             LOW-EinsPrungadresse-1 (0=Tabellenende)
```

Bild 4.1.3 (Teil 2)

PROGRAMM-EINZELHEITEN

1.) Name: VC-COMMAND

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Basic-Loader

4.) Anzahl der Bytes
des Basic-Loaders: 1801, 2532 mit DATA-Zeilen in Bild 4.1.2
des M-PGMs: 122

5.) Benutzte Variablen im Basic-Loader: s. Kap. 5.2
zusätzlich:

CM$	Befehlswort
ES	Einsprungadresse
TL	Tabellenlänge
LE	Länge von CM$
TB	Anfangsadresse des momentan abzuspeichernden Befehls in der Tabelle

6.) Listings
des Basic-Loaders: Bild 4.1.1 (+ Bild 4.1.2 als Beispiel)
des M-PGMs: Bild 4.1.3 (Teil 1 u. Teil 2)

7a) Erläuterungen zum Basic-Loader:

Es wird hier im Prinzip der Basic-Loader aus Kap 5.2 verwendet. Es wird deswegen nur auf die hinzugekommenen Zeilen eingegangen:

Zeilen 500 - 510:
Es wird der Speicherplatz berechnet, der zur Abspeicherung der in den DATA-Zeilen aufgelisteten Befehle benötigt wird. Dabei werden zu jedem Befehlswort 3 Byte hinzugezählt: eine 0 wird nämlich als Endekennzeichen des jeweiligen Befehls benutzt und die Einsprungadresse nehmen 2 weitere Bytes ein. Dieses Unterprogramm wird in Zeile 80 aufgerufen, bevor das M-PGM VC-COMMAND abgespeichert wird. Falls es am Ende des RAM-Bereichs generiert werden soll, muß verständlicherweise erst die Gesamtlänge von VC-COMMAND = 122 Bytes + Befehlstabelle bekannt sein. Die Gesamtlänge geht dann in Zeile 100 in die Berechnung der Anfangsadresse des M-PGMs ein.

Zeilen 600 - 650:
Der Aufruf dieses Unterprogramms erfolgt in Zeile 240, nachdem VC-COMMAND ohne Befehlstabelle generiert worden ist. Der Rest, also die Befehlstabelle, wird mittels diesem UP abgespeichert, und zwar in der Reihenfolge:
Befehlswort,0,Einsprungadresse HIGH,Einsprungadresse LOW
Als Endekennzeichen der gesamten Tabelle wird nochmals eine 0 angefügt.

Zeilen 1000 - 1999:
Hier befinden sich die DATA-Zeilen, welche die Befehlsworte und die entsprechenden Einsprungadressen enthalten. Das "Klammeraffen"-Symbol muß diese Liste als Endekennzeichen abschließen.

7b) Erläuterungen zum M-PGM sind implizit im Listing (Bild 4.1.3) enthalten.

4.2 NEUE FUNKTIONEN: KEEP, FETCH, VIEW, DISCARD, DELETE

Sie werden vielleicht schon einmal etwas über die Funktionen APPEND und MERGE gehört haben. Dies sind Funktionen, die das Zusammenfügen von Programmen bzw. Programmteilen ermöglichen. Allerdings weisen die bisher veröffentlichten Realisierungen dieser Funktionen einen Nachteil auf, so daß die Nutzungsmöglichkeiten erheblich eingeschränkt werden: sie erlauben nämlich eine Aneinanderkettung nur dann, wenn die Zeilennummern des jeweils hinzukommenden Programmteils größer sind als diejenigen des bereits im Arbeitsspeicher befindlichen Programms.

Für so manche Erfordernisse ist diese Einschränkung möglicherweise vertretbar. Eine viel elegantere Lösung jedoch wäre dann gegeben, wenn man mithilfe eines universell einsetzbaren Werkzeugs Programme beliebiger Zeilennumerierung ineinander verzahnen könnte. Das hier angebotene Programm VC-KEEP liefert dafür die Lösung. Dieses Maschinenprogramm (s. Bild 4.2.3) umfaßt zwar 879 Bytes, beinhaltet jedoch immerhin 5 Funktionen, die die Arbeit eines Basic-Programmierers um einiges komfortabler machen können.

Nicht nur das Mischen von Programmen nämlich ist mit VC-KEEP möglich, Sie haben damit auch die Möglichkeit, Programme vorübergehend per Funktion KEEP ans RAM-Ende "wegzulegen", wenn sie nicht benötigt werden. Jedes dieser weggespeicherten Programme kann man nach Belieben per FETCH-Funktion wieder in den Basic-Arbeitsspeicher zurückrufen und dann erneut benutzen. Ein andauerndes Hin und Her mit der Datassette entfällt hier also.

Die Funktion VIEW zeigt die Anfangsadressen sämtlicher am RAM-Ende vorhandenen M-PGMe an, bei dort abgelegten Programmen darüberhinaus noch deren Namen. VIEW ist hier gewissermaßen das, was das Abrufen der Directory bei der Diskette ist. Mit DISCARD wird der Speicherraum, den ein Programm am RAM-Ende einnimmt, wieder freigegeben. Die Funktion von DELETE ist den meisten von Ihnen wahrscheinlich schon geläufig. Sie dient dem schnellen Löschen von beliebig langen Teilen eines im Arbeitsspeicher vorhandenen Programms.

```
10 REM VC-KEEP
20 REM ZUSAETZL. FKT.:KEEP,FETCH,VIEW,DISCARD,DELETE
30 L=879
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>113065THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"AUFRUF DER FUNKTIONEN:"
270 PRINT"KEEP:"
280 PRINTCHR$(18)"SYS"SDCHR$(157)" "CHR$(34)"(NAME)"CHR$(34);
285 PRINT",(ZN)(-)(ZN)"CHR$(146):PRINT
290 PRINT"VIEW:"
300 PRINTCHR$(18)"SYS"SD+3CHR$(146):PRINT
310 PRINT"FETCH:"
320 PRINTCHR$(18)"SYS"SD+6CHR$(157)" "CHR$(34)"(NAME)"CHR$(34)CHR$(146):PRINT
330 PRINT"DISCARD:"
340 PRINTCHR$(18)"SYS"SD+9CHR$(157)" "CHR$(34)"(NAME)"CHR$(34)CHR$(146):PRINT
350 PRINT"DELETE:"
360 PRINTCHR$(18)"SYS"SD+12CHR$(157)",(ZN)(-)(ZN)"CHR$(146)
370 PRINT"ZN=ZEILENNUM.,()=OPT.":PRINT:PRINT
380 PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
1000 DATA76,L15,H,76,L657,H,76,L809,H,76,L835,H,169,128,44,169
1010 DATA64,141,66,3,32,L225,H,32,L234,H,44,66,3,48,3,76
1020 DATAL350,H,162,L127,160,H,32,L253,H,165,122,133,166,165,123,133
1030 DATA167,162,L71,160,H,32,L264,H,32,121,0,240,7,201,44,208
1040 DATA3,32,115,0,76,156,198,133,215,72,138,72,152,72,165,215
1050 DATA201,10,240,11,44,66,3,48,3,76,L307,H,76,L289,H,32
1060 DATAL271,H,169,196,205,3,3,240,7,162,98,32,L262,H,208,8
1070 DATA32,L260,H,169,13,32,122,242,104,168,104,170,104,24,96,32
1080 DATAL249,H,165,166,133,122,165,167,133,123,44,66,3,48,3,76
1090 DATAL481,H,32,L333,H,32,L229,H,32,L866,H,32,L164,H,32,95
1100 DATA229,76,L49,H,162,L176,160,H,120,142,20,3,140,21,3,96
1110 DATA32,L220,H,240,10,32,L201,H,176,12,32,L214,H,208,246,162
1120 DATA191,160,234,32,L168,H,76,191,234,166,198,236,137,2,176,5
1130 DATA157,119,2,230,198,96,230,166,208,2,230,167,160,0,177,166
1140 DATA96,169,0,160,0,133,163,132,164,96,169,0,141,68,3,141
1150 DATA67,3,141,65,3,141,64,3,96,162,131,160,196,142,2,3
```

Bild 4.2.1 (Teil 1)

```
1160 DATA140,3,3,96,162,122,160,242,142,38,3,140,39,3,96,169
1170 DATA0,44,65,3,16,4,160,0,145,163,230,163,208,2,230,164
1180 DATA96,44,64,3,48,9,201,13,208,17,206,64,3,48,4,201
1190 DATA32,240,5,32,L273,H,208,3,238,64,3,76,L120,H,162,80
1200 DATA160,3,96,134,187,132,188,96,134,170,132,171,96,56,165,55
1210 DATA229,163,133,166,170,165,56,229,164,133,167,168,138,96,32,L563
1220 DATAH,32,L619,H,32,L227,H,32,L318,H,32,L328,H,32,L428,H
1230 DATA165,183,24,105,10,133,251,32,L227,H,32,L333,H,32,L328,H
1240 DATA32,L464,H,32,L318,H,32,L323,H,165,183,32,L227,H,32,L428
1250 DATAH,165,183,32,L444,H,169,208,32,L444,H,169,208,32,L444,H
1260 DATA32,L444,H,32,L444,H,32,L225,H,76,L34,H,165,163,208,2
1270 DATA198,164,198,163,36,164,48,23,160,0,177,187,145,170,230,187
1280 DATA208,2,230,188,230,170,208,2,230,171,44,65,3,16,221,96
1290 DATA56,165,166,170,229,45,165,167,168,229,46,176,3,76,L873,H
1300 DATA96,206,65,3,165,251,24,101,163,133,163,169,0,101,164,133
1310 DATA164,32,L333,H,32,L229,H,32,L464,H,32,L542,H,165,166,160
1320 DATA0,32,L444,H,165,167,32,L444,H,169,197,32,L444,H,169,206
1330 DATA32,L444,H,169,196,32,L444,H,76,L49,H,32,L323,H,134,55
1340 DATA132,56,134,51,132,52,165,45,133,47,133,49,165,46,133,48
1350 DATA133,50,96,32,121,0,208,3,76,8,207,201,44,208,3,32
1360 DATA115,0,32,84,226,134,193,170,208,4,162,8,208,6,201,129
1370 DATA144,5,162,23,76,58,196,133,251,132,194,32,L666,H,165,251
1380 DATA133,183,166,193,134,187,164,194,132,188,96,44,68,3,16,14
1390 DATA162,2,208,7,44,68,3,48,5,162,4,76,58,196,96,162
1400 DATA1,165,187,208,2,198,188,198,187,160,0,177,187,202,208,241
1410 DATA96,169,13,32,122,242,56,110,67,3,32,L225,H,174,131,2
1420 DATA172,132,2,32,L539,H,169,0,133,188,32,L639,H,201,136,208
1430 DATA113,32,L639,H,201,206,208,112,32,L639,H,201,197,208,105,32
1440 DATAL639,H,72,32,L639,H,170,104,168,32,L328,H,162,3,32,L641
1450 DATAH,201,208,208,16,32,L639,H,201,208,208,9,32,L639,H,138
1460 DATA183,170,32,L641,H,44,67,3,16,30,166,170,165,171,32,205
1470 DATA221,169,32,32,122,242,165,183,240,3,32,89,246,169,13,32
1480 DATA122,242,166,170,164,171,208,155,164,183,196,251,208,244,136,177
1490 DATA193,209,187,208,237,136,16,247,206,68,3,32,L333,H,165,170
1500 DATA133,168,165,171,133,169,208,218,96,32,L225,H,32,L234,H,32
1510 DATAL563,H,32,L628,H,165,168,133,166,165,169,133,167,32,95,229
1520 DATA76,L164,H,32,L225,H,32,L234,H,32,L563,H,32,L628,H,165
1530 DATA166,208,2,198,167,198,166,160,0,152,145,166,32,L225,H,76
1540 DATAL666,H,206,65,3,32,L464,H,96,32,L260,H,76,53,196
```

Bild 4.2.1 (Teil 2)

Das Maschinenprogramm VC-KEEP wird je nach Ihrem Wunsch in einem beliebigen Speicherbereich generiert. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen Überschreiben geschutzt. Weiterhin sind die in VC-KEEP enhaltenen Funktionen KEEP, VIEW, FETCH, DISCARD und DELETE koppelbar mit von Ihnen frei definierbaren Befehlen (s. Kap. 4.1).

BEDIENUNGSHINWEISE

VC-KEEP ist ein Maschinen-Programm, das in den im Kapitel 5.2 beschriebenen Basic-Loader eingebunden wurde. Detailinformationen können dort in Erfahrung gebracht werden. Der Basic-Loader wird normal mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das M-PGM generiert werden soll. Bei Eingabe von "E" wird es ans Basic-RAM-Ende gespeichert, wobei eventuell schon andere vorhandene M-PGMe in keinster Weise angetastet werden.

Nachdem die Eingabe einer Anfangsadresse oder von "E" durch RETURN bestatigt worden ist, werden Ihnen auf dem Bildschirm die Aufruf-Adressen fur die 5 Funktionen angezeigt. Zur besseren Erläuterung gehen wir von einem Beispiel aus. Sie lernen damit prinzipiell den Gebrauch der Funktionen kennen und können die Regeln auf andere Anwendungsfälle übertragen.

Z.B. besitzen Sie einen VC-20 mit einer 16 KByte-Speichererweiterung. Nach Starten des VC-KEEP-Basic-Loaders mit RUN entscheiden Sie sich dafur, das M-PGM am Ende des Basic-RAM generieren zu lassen. Sie geben also ein: E (Return). Daraufhin werden folgende Hinweise gegeben:

```
KEEP:
SYS 23690,"(Name)",(ZN)(-)(ZN)

VIEW:
SYS 23693

FETCH:
SYS 23696,"(Name)"

DISCARD:
SYS 23699,"(Name)"

DELETE:
SYS 23702,(ZN)(-)(ZN)
```

Die Angabe der Zeilennummer (ZN) und des Bindestrichs (-) sind optional. Die angegebenen SYS-Anweisungen schreibt man sich am besten irgendwo auf, merken tut man sich die ja nicht.

Probieren wir jetzt gleich einmal die Funktion VIEW aus, indem wir SYS 23693 eingeben. Als Reaktion erscheint die Zahl 23690 auf dem Bildschirm. Das ist die Anfangsadresse des am Basic-RAM-Ende abgespeicherten M-PGMs VC-KEEP. Wenn übrigens noch andere M-PGMe am RAM-Ende vorhanden waren, würden die Anfangsadressen sämtlicher M-PGMe untereinander angezeigt werden, vorausgesetzt, sie wurden mit unserem speziellen Basic-Loader aus Kap. 5.2 generiert.

Die VIEW-Funktion beschränkt sich nicht nur darauf, alle diese Anfangsadressen aufzulisten, sondern übt noch eine andere wesentliche Aufgabe aus: falls es irgendwann einmal notwendig werden sollte, die RESET-Taste zu drücken, dann waren ja sämtliche M-PGMe am Basic-RAM-Ende vor Überschreiben ungeschützt, weil der Ende-Vektor (Adressen #55,56) reinitialisiert worden ist. VIEW muß in diesem Fall unmittelbar nach RESET aufgerufen werden, weil es die vorliegenden M-PGMe identifiziert und den Vektor #55,56 auf die Anfangsadresse des zuunterst liegenden M-PGMs ausrichtet. Aufgrund dieser Rettungsfähigkeit wird diese Funktion in den M-PGM-Kommentaren (Bild 4.2.3) auch als VIEW & RESCUE bezeichnet.

Zurück zu unserem Beispiel, wir geben jetzt die Programmzeile ein:

```
20 PRINT "IST"
```

und speichern sie mithilfe der KEEP-Funktion ans RAM-Ende:

```
SYS 23690,"ZEILE 20"
```

Der Name "ZEILE 20" ist beliebig. Ein Name ist jedoch notwendig, um diesen Programmteil - auch wenn es sich nur um eine einzige Zeile handelt - identifizierbar zu machen. Die Angabe der Von-Bis-Zeilennummern sparen wir uns, weil wir alle vorhandenen Programmzeilen - in diesem Fall gibt es nur die Zeile 20 - geKEEPed wünschen. Nebenbei bemerkt, wenn Sie die 5 oben angegebenen SYS-Einsprungadressen mit den Befehlsworten KEEP, VIEW, FETCH, DISCARD und DELETE (oder auch anderen Worten) über das Programm VC-COMMAND aus Kapitel 4.1 koppeln, dann könnten Sie in luxuriöser Weise eingeben:

```
KEEP "ZEILE 20"
```

Üben wir jetzt wieder die Funktion VIEW aus. Per SYS 23693 erhalten wir jetzt folgerichtig 2 Angaben auf dem Bildschirm:

```
23690
23656 ZEILE 20
```

Wir können daraus ableiten, daß die PGM-Zeile 20 ab Adresse 23656 nach oben hin abgespeichert wurde. Außerdem gibt die letzte mittels VIEW angezeigte Adresse auch gleichzeitig die augenblicklich für Basic gültige obere Grenzadresse an.

Als nächste Programmzeile werden wir

```
40 PRINT "KLEINES"
```

eingeben, welche wir ähnlich wie oben per

```
SYS 23690,"ZEILE 40",40
```

"weglegen". Die Angabe "40" hinter dem Namen muß durch ein Komma vom Namen getrennt werden und sagt aus, daß nur die Zeile 40 gespeichert werden soll. Die VIEW-Funktion hätte jetzt folgendes Resultat:

```
23690
23656 ZEILE 20
23618 ZEILE 40
```

Per NEW leeren wir nun den Basic-Arbeitsspeicher und tippen die folgenden Zeilen ein:

```
10 PRINT "DIES"
30 PRINT "EIN"
50 PRINT "DEMO-PROGRAMM"
```

Die FETCH-Funktion werden wir jetzt dazu verwenden, dieses Programm um die weggespeicherten Zeilen zu vervollständigen.

Nach Eingabe von:

```
SYS 23696,"ZEILE 20"
SYS 23696,"ZEILE 40"
```

und Programmstart per RUN können wir uns aufgrund der Ausgabe:

```
DIES
IST
EIN
KLEINES
DEMO-PROGRAMM
```

davon überzeugen, daß tatsächlich die abgespeicherten Zeilen 20 und 40 in der richtigen Reihenfolge in das vorherige Programm (Zeilen 10, 30, 50) eingeordnet wurden.

Da wir die "weggelegten" Programmzeilen 20 und 40 nicht mehr zurückrufen werden, geben wir den Speicherbereich per DISCARD-Funktion wieder frei. Hierbei ist zu beachten, daß bei Löschen eines Programmteils gleichzeitig sämtliche unterhalb diesem Teil liegenden andere Programmteile und M-PGMe ebenfalls gelöscht werden. Mit anderen Worten:

```
SYS 23699,"ZEILE 20"
```

würde nicht nur den Speicherplatz von "ZEILE 20", sondern auch von "ZEILE 40" freigeben, wovon Sie sich schnell per VIEW-Funktion überzeugen konnen.

Die Funktionen KEEP und DELETE können Sie auf einen beliebigen "Ausschnitt" Ihres Programms anwenden. Den Ausschnitt bezeichnen Sie in der gewohnten Weise wie bei LIST.

Wollen Sie also z.B. die Programmzeilen 10-30 unter dem Namen "1. TEIL" wegspeichern und die Zeilen 30 und 40 löschen, so geben Sie ein:

```
SYS 23690,"1. TEIL",-30
SYS 23702,30-40
```

Sie werden sich vielleicht schon gefragt haben, woran das M-PGM VC-KEEP die unter einem Namen abgespeicherten Basic-Programmzeilen von M-PGMen unterscheiden kann. In Kapitel 5.2 erfahren wir (Bild 5.2-2), daß an ein M-PGM 7 Bytes drangehangt werden, die die Existenz eines M-PGMs auch nach RESET erkennbar machen.

Bild 4.2.2

Währenddessen bei M-PGMen den ersten 2 Bytes dieses Anhangsels keine Bedeutung gegeben wurde, sieht dies bei Programmteilen anders aus (s. Bild 4.2.2). Hierbei werden den ersten zwei Bytes die Werte $D0 (ASCII-Code von "P" mit gesetztem Bit 7) gegeben. VC-KEEP kann bei Vorhandensein dieser Werte erkennen, daß es sich um Programmzeilen handelt und weiß darüberhinaus, daß deren Name und die Länge des Namens (NL) unmittelbar unterhalb dieser 2 Bytes gespeichert ist.

PROGRAMM-EINZELHEITEN

1.) Name: VC-KEEP

2.) Ausbaustufe: 3K-V oder >=8K-V

3.) Art des PGMs: Basic-Loader

4.) Anzahl der Bytes
des Basic-Loaders: 4585
des M-PGMs: 879

5.) Benutzte Variablen: s. Kap. 5.2

6.) Listings
des Basic-Loaders: Bild 4.2.1 (Teil 1 und 2)
des M-PGMs: Bild 4.2.3 (Teile 1 - 7)

7.) Erläuterungen zum Basic-Loader werden in Kap. 5.2 gegeben und zum M-PGM implizit im ausführlich kommentierten Listing (Bild 4.2.3).

```
2000 JMP $200F          Funktion: KEEP
2003 JMP $2291          Funktion: VIEW & RESCUE
2006 JMP $2329          Funktion: FETCH
2009 JMP $2343          Funktion: DISCARD
200C LDA #$80           Beginn Funktion: DELETE
200E BIT $40A9          Adr.: $200F LDA #$40
2011 STA $0342          Flag sPeichern fuer DELETE/KEEP
2014 JSR $20E1          $A3,$A4 = 00
2017 JSR $20EA          Flags auf Default setzen
201A BIT $0342          Weiche fuer DELETE/KEEP
201D BMI $2022          SPrung zur Fkt. DELETE
201F JMP $215E          SPrung zur Fkt. KEEP
2022 LDX #$7F           BASIC-Warteschleifen-Vektor verbiegen
2024 LDY #$20
2026 JSR $20FD
2029 LDA $7A            Scanner-Vektor retten
202B STA $A6                    nach $A6,$A7
202D LDA $7B
202F STA $A7
2031 LDX #$47           OUTPUT-Vektor verbiegen
2033 LDY #$20
2035 JSR $2108
2038 JSR $0079          Von - Bis  Zeilennummer einlesen
203B BEQ $2044
203D CMP #$2C
203F BNE $2044
2041 JSR $0073
2044 JMP $C69C          SPrung zur LIST-Routine
2047 STA $D7
2049 PHA                    Register A,X,Y retten
204A TXA
204B PHA
204C TYA
204D PHA
204E LDA $D7
2050 CMP #$0A           Bereich fuer DELETE bzw. KEEP  zuende
2052 BEQ $205F
2054 BIT $0342          Weiche fuer DELETE/KEEP
2057 BMI $205C          Fortsetzung DELETE
2059 JMP $2133          SPrung zur Fortsetzung d. Fkt. KEEP
205C JMP $2121          SPrung zur Fortsetzung d. Fkt. DELETE
205F JSR $210F          $00 als Endekennzeichen absPeichern
2062 LDA #$C4
2064 CMP $0303
2067 BEQ $2070          SPrung, wenn AbsPeicherung fertig
2069 LDX #$62           OUTPUT-Vektor auf $F262 = RTS stellen
206B JSR $2106                  entsPricht: Ausgabe blockieren
206E BNE $2078          SPrung: immer
2070 JSR $2104          OUTPUT-Vektor wiederherstellen
2073 LDA #$0D           CR
2075 JSR $F27A             Ausgeben
2078 PLA
2079 TAY                    Register A,X,Y zurueckholen
207A PLA
207B TAX
207C PLA
207D CLC
207E RTS
```

Bild 4.2.3 (Teil 1)

```
207F JSR $20F9          BASIC-Warteschleifenv. wiederherstellen
2082 LDA $A6
2084 STA $7A                Scanner-Vektor zurueckstellen
2086 LDA $A7
2088 STA $7B
208A BIT $0342          Weiche fuer DELETE/KEEP
208D BMI $2092          Fortsetzung DELETE
208F JMP $21E1          Sprung zur Fortsetzung d. Fkt. KEEP
2092 JSR $214D          Neues BASIC-Ende nach $A6,$A7 und X-A,Y
2095 JSR $20E5          A,Y nach $A3,$A4
2098 JSR $2362          Memory-Check
209B JSR $20A4          Tastaturpuffer-Eingabe freigeben
209E JSR $E55F          CLR HOME
20A1 JMP $2031          zum Abspeichern
20A4 LDX #$B0           Interrupt-Routine umleiten
20A6 LDY #$20
20A8 SEI
20A9 STX $0314
20AC STY $0315
20AF RTS
20B0 JSR $20DC          Momentanes Zeichen holen
20B3 BEQ $20BF          Sprung, wenn Text zuende
20B5 JSR $20C9          Zeichen nach Tastaturpuffer (Versuch)
20B8 BCS $20C6          Sprung, wenn Tastaturpuffer voll
20BA JSR $20D6          Naechstes Zeichen holen
20BD BNE $20B5          Sprung, wenn Text nicht zuende
20BF LDX #$BF           Interrupt-Rout. auf alten Stand bringen
20C1 LDY #$EA
20C3 JSR $20A8
20C6 JMP $EABF          Ende der Interrupt-Routinen-Umleitung
20C9 LDX $C6            Wieviele Zeichen im Tastaturpuffer?
20CB CPX $0289          mit maximal erlaubter Anzahl vergleichen
20CE BCS $20D5          Sprung raus, wenn Puffer voll
20D0 STA $0277,X        Zeichen in den Puffer ablegen
20D3 INC $C6
20D5 RTS
20D6 INC $A6            $A6,$A7 = ($A6,$A7)+1
20D8 BNE $20DC
20DA INC $A7
20DC LDY #$00
20DE LDA ($A6),Y
20E0 RTS
20E1 LDA #$00
20E3 LDY #$00               Vektor $A3,$A4 setzen
20E5 STA $A3
20E7 STY $A4
20E9 RTS
20EA LDA #$00           Flags auf Default setzen:
20EC STA $0344          File-Name vorhanden / nicht vorhanden
20EF STA $0343          File-N. suchen / Adr. + File-N. ausgeben
20F2 STA $0341          Zeichen zaehlen / abspeichern
20F5 STA $0340          Zeichen =  /  <>Zeilennummernziffer
20F8 RTS
20F9 LDX #$83           BASIC-Warteschleifen-Vektor setzen
20FB LDY #$C4
20FD STX $0302
2100 STY $0303
2103 RTS
```

Bild 4.2.3 (Teil 2)

```
2104 LDX #$7A
2106 LDY #$F2                    OUTPUT-Vektor setzen
2108 STX $0326
210B STY $0327
210E RTS
------------------------------------------------------------------------
210F LDA #$00                Zaehl- bzw. AbsPeicher-UP
2111 BIT $0341               Weiche fuer Zaehlen/AbsPeichern
2114 BPL $211A               SPrun9 zum Zaehlen
2116 LDY #$00
2118 STA ($A3),Y             AbsPeichern
211A INC $A3
211C BNE $2120
211E INC $A4
2120 RTS
------------------------------------------------------------------------
2121 BIT $0340
2124 BMI $212F                   Zeilennummern
2126 CMP #$0D                         von Zeileninhalt trennen
2128 BNE $213B
212A DEC $0340               Zeilennummer be9innt
212D BMI $2133
212F CMP #$20
2131 BEQ $2138
2133 JSR $2111               Zaehlen bzw. AbsPeichern
2136 BNE $213B
2138 INC $0340               Zeilennummer-Ende
213B JMP $2078               zum OUTPUT-Umleitun9sende
------------------------------------------------------------------------
213E LDX #$50
2140 LDY #$03                    KassettenPuffer-Vektor nach X,Y
2142 RTS
------------------------------------------------------------------------
2143 STX $BB
2145 STY $BC                     $BB,$BC setzen
2147 RTS
------------------------------------------------------------------------
2148 STX $AA
214A STY $AB                     $AA,$AB setzen
214C RTS
------------------------------------------------------------------------
214D SEC
214E LDA $37                     BASIC-Ende berechnen
2150 SBC $A3
2152 STA $A6                          und nach $A6,$A7 und
2154 TAX
2155 LDA $38                                A-X,Y
2157 SBC $A4
2159 STA $A7
215B TAY
215C TXA
215D RTS
------------------------------------------------------------------------
215E JSR $2233               Name einlesen
2161 JSR $226B               Name auf "nicht vorhanden" ueberPruefen
2164 JSR $20E3               Laen9e des Namens nach $A3,$A4
2167 JSR $213E               KassettenPuffer-Vektor
216A JSR $2148                                        nach $AA,$AB
216D JSR $21AC               Name nach KassettenPuffer retten
2170 LDA $B7                 Namen-Laen9e
2172 CLC
2173 ADC #$0A                10 addieren
2175 STA $FB                 Namen-Laen9e + 10  nach $FB retten
```

Bild 4.2.3 (Teil 3)

```
2177 JSR $20E3          Namen-Laenge + 10 nach $A3,$A4
217A JSR $214D          neues BASIC-Ende berechnen
217D JSR $2148          BASIC-Ende nach $AA,$AB
2180 JSR $21D0          OUT OF MEMORY ???
2183 JSR $213E          KassettenPuffer-Vektor
2186 JSR $2143                                  nach $BB,$BC
2189 LDA $B7            Namen-Laenge
218B JSR $20E3                      nach $A3,$A4 (als Zaehler)
218E JSR $21AC          Name ans BASIC-Ende bringen
2191 LDA $B7            Namen-Laenge
2193 JSR $21BC                      hinter Namen abspeichern
2196 LDA #$D0           "$D0"
2198 JSR $21BC                dahinter abspeichern
219B LDA #$D0           "$D0"
219D JSR $21BC                dahinter abspeichern
21A0 JSR $21BC          Reserve-Byte dahinter abspeichern
21A3 JSR $21BC          Reserve-Byte dahinter abspeichern
21A6 JSR $20E1          $A3,$A4 auf 0 setzen
21A9 JMP $2022          Sprung zum Zaehlen der PGM-Bytes
-------------------------------------------------------------
21AC LDA $A3
21AE BNE $21B2              UnterProgramm
21B0 DEC $A4                      zum Kopieren Speicherzellen
21B2 DEC $A3                      $BB,$BC =  Quell-Vektor
21B4 BIT $A4                      $AA,$AB =  Ziel-Vektor
21B6 BMI $21CF
21B8 LDY #$00
21BA LDA ($BB),Y
21BC STA ($AA),Y
21BE INC $BB            $BB,$BC = ($BB,$BC) + 1
21C0 BNE $21C4
21C2 INC $BC
21C4 INC $AA            $AA,$AB = ($AA,$AB) + 1
21C6 BNE $21CA
21C8 INC $AB
21CA BIT $0341
21CD BPL $21AC
21CF RTS
-------------------------------------------------------------
21D0 SEC
21D1 LDA $A6                neuen BASIC-Ende-Vektor mit
21D3 TAX
21D4 SBC $2D                  Variablenanfang-Vektor vergleichen
21D6 LDA $A7
21D8 TAY
21D9 SBC $2E
21DB BCS $21E0
21DD JMP $2369          Sprung zur Ausgabe: "OUT OF MEMORY"
21E0 RTS
-------------------------------------------------------------
21E1 DEC $0341          Freigabe zur Abspeicherung
21E4 LDA $FB            Namen-Laenge + 10
21E6 CLC                    zur Anzahl
21E7 ADC $A3                      der PGM-Bytes
21E9 STA $A3                       hinzuzaehlen
21EB LDA #$00
21ED ADC $A4
21EF STA $A4
21F1 JSR $214D          neues BASIC-Ende berechnen
21F4 JSR $20E5              und nach $A3,$A4
21F7 JSR $21D0          OUT OF MEMORY ???
21FA JSR $221E          neuen BASIC-Ende-Vektor setzen
```

Bild 4.2.3 (Teil 4)

```
21FD LDA $A6              LOW-Byte von BASIC-Ende
21FF LDY #$00
2201 JSR $21BC                hinter Namen abspeichern
2204 LDA $A7              HIGH-Byte von BASIC-Ende
2206 JSR $21BC                dahinter abspeichern
2209 LDA #$C5             "E"
220B JSR $21BC                abspeichern
220E LDA #$CE             "N"
2210 JSR $21BC                abspeichern
2213 LDA #$C4             "D"
2215 JSR $21BC                abspeichern
2218 JMP $2031            Sprung zum Abspeichern
-------------------------------------------------------
221B JSR $2143            X,Y nach $BB,$BC
221E STX $37
2220 STY $38                  neues BASIC-Ende = X,Y
2222 STX $33
2224 STY $34                  + CLR-Funktion
2226 LDA $2D
2228 STA $2F
222A STA $31
222C LDA $2E
222E STA $30
2230 STA $32
2232 RTS
-------------------------------------------------------
2233 JSR $0079            Momentanes Zeichen holen
2236 BNE $223B
2238 JMP $CF08            "SYNTAX ERROR", wenn kein Zeichen mehr
223B CMP #$2C             Zeichen = "," ?
223D BNE $2242            Sprung, wenn nein
223F JSR $0073            Naechstes Zeichen holen
2242 JSR $E254            String (Namen) einlesen
2245 STX $C1              LOW-Anfangsadresse vom Namen nach $C1
2247 TAX                  Z-Flag fuer Namen-Laenge setzen
2248 BNE $224E            Sprung, wenn Laenge <>0
224A LDX #$08             zur Ausgabe:
224C BNE $2254                    "MISSING FILE NAME"
224E CMP #$81             Laenge > 128 ?
2250 BCC $2257            Sprung, wenn nein
2252 LDX #$17             zur Ausgabe:
2254 JMP $C43A                    "STRING TOO LONG"
2257 STA $FB              Laenge nach $FB
2259 STY $C2              HIGH-Anfangsadresse vom Namen nach $C2
225B JSR $229A            Pruefen, ob File-Name schon vorhanden
225E LDA $FB              Namen-Deskriptor $FB,$C1,$C2
2260 STA $B7                      nach $B7,$BB,$BC
2262 LDX $C1
2264 STX $BB
2266 LDY $C2
2268 STY $BC
226A RTS
-------------------------------------------------------
226B BIT $0344            Flag fuer Name nicht/-- gefunden
226E BPL $227E            Sprung, wenn nicht gefunden
2270 LDX #$02             zur Ausgabe:
2272 BNE $227B                    "FILE OPEN"
2274 BIT $0344            siehe $226B
2277 BMI $227E            Sprung, wenn Name gefunden
2279 LDX #$04             zur Ausgabe:
227B JMP $C43A                    "FILE NOT FOUND"
227E RTS
-------------------------------------------------------
```

Bild 4.2.3 (Teil 5)

```
227F LDX ##$01          $BB,$BC = ($BB,$BC) - 1
2281 LDA $BB            $BB,$BC = ($BB,$BC) - X
2283 BNE $2287
2285 DEC $BC
2287 DEC $BB                und
2289 LDY ##$00
228B LDA ($BB),Y        Zeichen ($BB),0 holen
228D DEX
228E BNE $2281
2290 RTS
2291 LDA ##$0D          CR
2293 JSR $F27A             ausgeben
2296 SEC
2297 ROR $0343          FLAG auf "Ausgabe" schalten
229A JSR $20E1          $A3,$A4 = 0
229D LDX $0283          RAM-Ende-Vektor
22A0 LDY $0284              nach
22A3 JSR $221B                  BASIC-Ende-Vektor und $BB,$BC
22A6 LDA ##$00          Default-Namenlaenge = 0
22A8 STA $B7                                    nach $B7
22AA JSR $227F          $BB,$BC = ($BB,$BC)-1 und Zeichen holen
22AD CMP ##$C4          Zeichen = "D"
22AF BNE $2328          RTS, wenn nein
22B1 JSR $227F          $BB,$BC = ($BB,$BC)-1 und Zeichen holen
22B4 CMP ##$CE          Zeichen = "N"
22B6 BNE $2328          RTS, wenn nein
22B8 JSR $227F          $BB,$BC = ($BB,$BC)-1 und Zeichen holen
22BB CMP ##$C5          Zeichen = "E"
22BD BNE $2328          RTS, wenn nein
22BF JSR $227F          HIGH-Byte von BASIC-Ende holen
22C2 PHA                    und auf Stack
22C3 JSR $227F          LOW-Byte von BASIC-Ende holen
22C6 TAX                    und nach X
22C7 PLA                HIGH-Byte
22C8 TAY                        nach Y
22C9 JSR $2148          X,Y nach $AA,$AB
22CC LDX ##$03          $BB,$BC = ($BB,$BC)-3
22CE JSR $2281              und Zeichen holen
22D1 CMP ##$D0          Zeichen = "$D0"
22D3 BNE $22E5          Sprung, wenn nein
22D5 JSR $227F          $BB,$BC = ($BB,$BC)-1 und Zeichen holen
22D8 CMP ##$D0          Zeichen = "$D0"
22DA BNE $22E5          Sprung, wenn nein
22DC JSR $227F          $BB,$BC = ($BB,$BC)-1, Namenlaenge holen
22DF STA $B7            Namenlaenge nach $B7
22E1 TAX                    und nach X
22E2 JSR $2281          $BB,$BC = ($BB,$BC)-X
22E5 BIT $0343          Weiche fuer Suchen/Ausgabe
22E8 BPL $2308          Sprung zum Suchen
22EA LDX $AA            Anfangsadresse
22EC LDA $AB                            dezimal
22EE JSR $DDCD                                  ausgeben
22F1 LDA ##$20          SPACE
22F3 JSR $F27A              ausgeben
```

Bild 4.2.3 (Teil 6)

```
22F6 LDA $B7            Namen-Laenge
22F8 BEQ $22FD          Sprung, wenn Laenge = 0
22FA JSR $F659          Ausgabe des Namens
22FD LDA #$0D           CR
22FF JSR $F27A             ausgeben
2302 LDX $AA            Anfangsadresse (Link) nach X,Y
2304 LDY $AB
2306 BNE $22A3          Sprung: immer, zum naechsten Namen
2308 LDY $B7            Vergleich
230A CPY $FB                    der Namen-Laengen
230C BNE $2302          Sprung zum Weitersuchen, wenn ungleich
230E DEY
230F LDA ($C1),Y        Vergleich der
2311 CMP ($BB),Y                  Namen
2313 BNE $2302          Sprung zum Weitersuchen, wenn ungleich
2315 DEY
2316 BPL $230F
2318 DEC $0344          Name gefunden
231B JSR $214D          aktuelles BASIC-Ende nach $A6,$A7 retten
231E LDA $AA            File-Anfang-Link nach $A8,$A9 retten
2320 STA $A8
2322 LDA $AB
2324 STA $A9
2326 BNE $2302          Sprung: immer
2328 RTS
2329 JSR $20E1          $A3,$A4 = 0
232C JSR $20EA          Flags initialisieren
232F JSR $2233          File-Namen einlesen
2332 JSR $2274          Pruefen, ob Name vorhanden
2335 LDA $A8            File-Anfang-Vektor nach $A6,$A7
2337 STA $A6
2339 LDA $A9
233B STA $A7
233D JSR $E55F          CLR HOME
2340 JMP $20A4          zur TastaturPuffer-Eingabe
2343 JSR $20E1          $A3,$A4 = 0
2346 JSR $20EA          Flags initialisieren
2349 JSR $2233          File-Namen einlesen
234C JSR $2274          Pruefen, ob Name vorhanden
234F LDA $A6            Am
2351 BNE $2355             File-Ende
2353 DEC $A7                     eine
2355 DEC $A6                          0 setzen
2357 LDY #$00
2359 TYA
235A STA ($A6),Y
235C JSR $20E1          $A3,$A4 = 0
235F JMP $229A          VIEW & RESCUE
2362 DEC $0341          Flag auf "AbsPeichern" setzen
2365 JSR $21D0          OUT OF MEMORY ???
2368 RTS
2369 JSR $2104          OUTPUT-Vektor wiederherstellen
236C JMP $C435          Ausgabe: "OUT OF MEMORY"
```

Bild 4.2.3 (Teil 7)

4.3 FIND

FIND ist eine für den Programmierer sehr hilfreiche Funktion. Nicht selten kommt es vor, daß bestimmte Textteile, Zahlen oder Befehlsworte im Programm gesucht werden, wenn man einige Änderungen daran vornehmen möchte. FIND spürt diese Stellen im Programm auf.

Da das VC-20-Betriebssystem diese Funktion nicht in sich birgt, muß diese durch ein eigenes Programm realisiert werden. Das Maschinenprogramm VC-FIND (s. Bild 4.3.2) wird je nach Ihrem Wunsch in einem beliebigen Speicherbereich generiert. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es von selbst gegen Überschreiben geschützt. Weiterhin ist es koppelbar mit von Ihnen frei definierbaren Befehlen (s. Kap. 4.1).

```
10 REM VC-FIND
20 REM AUFSUCHEN VON BELIEB. ZEICHENKETTEN
30 L=103
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA:CH=CH+A
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANFANGSADRESSE AENDERN":STOP
230 IFCH<>15550THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(157)":(ZEICHENKETTE)"CHR$(146):PRINT
280 PRINT:PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
1000 DATA32,115,0,165,43,133,163,165,44,133,164,160,0,177,163,133
1010 DATA166,200,17,163,240,78,177,163,133,167,200,177,163,133,168,200
1020 DATA177,163,133,169,200,177,163,240,49,162,0,193,122,240,4,200
1030 DATA234,208,242,192,0,240,10,136,230,163,208,2,230,164,234,208
1040 DATA242,160,0,177,122,240,7,209,163,208,218,200,208,245,169,13
1050 DATA32,122,242,166,168,165,169,32,205,221,165,166,133,163,165,167
1060 DATA133,164,208,167,76,116,196
```

Bild 4.3.1

BEDIENUNGSHINWEISE

Das eigentliche Programm ist ein Maschinen-Programm, das in den im Kapitel 5.2 beschriebenen Basic-Loader (s. Bild 4.3.1) eingebunden wurde. Detailinformation bezgl. diesem kann dort in Erfahrung gebracht werden. Der Basic-Loader wird normal mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das M-PGM generiert werden soll. Bei Eingabe von "E" wird es ans Basic-RAM-Ende gespeichert, wobei eventuell schon andere vorhandene M-PGMe in keinster Weise angetastet werden.

Nachdem Sie sich für einen Standort des Maschinenprogramms entschieden haben, drücken Sie RETURN, worauf Sie eine Anzeige auf dem Bildschirm erhalten, die Ihnen Auskunft darüber gibt, wie Sie FIND aufrufen und benutzen.

Beispielsweise besitzen Sie einen VC-20 in Grundversion und lassen VC-FIND am Ende des Basic-RAM generieren. In diesem Fall rufen Sie FIND per:

```
M-PGM-AUFRUF MIT:
SYS 7570:(Suchwort)      (Return)
```

Suchen Sie z.B. das Befehlswort POKE, dann wäre einzugeben:

```
SYS 7570:POKE            (Return)
```

Bei der Suche nach Befehlsworten können auch die gebräuchlichen Kürzel benutzt werden, also für POKE: Taste "P" und SHIFT- + "O"-Taste.

Als Resultat werden sämtliche Zeilennummern auf den Bildschirm ausgegeben, die das Suchwort enthalten.

PROGRAMM-EINZELHEITEN

1.) Name: VC-FIND

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Basic-Loader

```
2000 JSR $0073          Hole Zeichen nach DoPPelPunkt
2003 LDA $2B            BASIC-Anfang-Vektor nach $A3,$A4
2005 STA $A3
2007 LDA $2C
2009 STA $A4
200B LDY #$00           Vektor $A6,$A7 wird auf
200D LDA ($A3),Y            Anfangsadresse der Folgezeile
200F STA $A6                                    gerichtet
2011 INY
2012 ORA ($A3),Y
2014 BEQ $2064          Adresse =0 => fertig
2016 LDA ($A3),Y
2018 STA $A7
201A INY
201B LDA ($A3),Y        Momentane Zeilennummer
201D STA $A8                nach $A8,$A9
201F INY
2020 LDA ($A3),Y
2022 STA $A9
2024 INY
2025 LDA ($A3),Y        Zeilenzeichen
2027 BEQ $205A          SPrung bei Zeileende
2029 LDX #$00           Zeilenzeichen mit
202B CMP ($7A,X)            Suchwortanfangszeichen vergleichen
202D BEQ $2033          Suchwortanfangszeichen gleich
202F INY
2030 NOP
2031 BNE $2025          SPrung: immer
2033 CPY #$00
2035 BEQ $2041          Addition $A3,$A4=($A3,$A4)+Y  fertig
2037 DEY
2038 INC $A3
203A BNE $203E
203C INC $A4
203E NOP
203F BNE $2033
2041 LDY #$00
2043 LDA ($7A),Y        Suchwortzeichen
2045 BEQ $204E          Suchwort in Zeile gefunden
2047 CMP ($A3),Y        Zeilenzeichen
2049 BNE $2025          ungleiche Zeichen => weitersuchen
204B INY
204C BNE $2043          SPrung: immer
204E LDA #$0D           CR
2050 JSR $F27A              ausgeben
2053 LDX $A8            Zeilennummer
2055 LDA $A9                        ausgeben
2057 JSR $DDCD
205A LDA $A6            $A3,$A4 = ($A6,$A7)
205C STA $A3
205E LDA $A7
2060 STA $A4
2062 BNE $200B
2064 JMP $C474          zum READY-Modus
```

Bild 4.3.2

4.) Anzahl der Bytes
des Basic-Loaders: 1207
des M-PGMs: 103

5a) Benutzte Variablen im Basic-Loader: s. Kap. 5.2

5b) Benutzte Adressen im M-PGM:

$A3,A4	Anfangsadresse der momentanen Zeile
$A6,A7	Anfangsadresse der Folgezeile
$A8,A9	Momentane Zeilennummer

6.) Listings
des Basic-Loaders: Bild 4.3.1
des M-PGMs: Bild 4.3.2

7.) Erläuterungen zum Basic-Loader werden in Kap. 5.2 gegeben und zum M-PGM implizit im Listing (Bild 4.3.2).

4.4 INTELLIGENTES BREAK/CONT

Eine gebräuchliche Methode zur Fehlersuche in Basic-Programmen ist es, an markanten Stellen STOP-Befehle einzufügen. Der Programmierer hat dadurch die Möglichkeit, die bis dahin errbeiteten Daten zu überprüfen, das Programm zu ändern und gegebenenfalls mit CONT weiterzuarbeiten. CONT funktioniert aber nur, wenn das Programm nicht geändert wurde, denn eine Programmänderung hat zur Folge, daß alle Variablen gelöscht und alle Programmzeiger zurückgesetzt werden.

Die hier vorgestellte Maschinensprache-Routine erlaubt es, Programme ohne Datenverlust zu editieren und danach mit CONT oder GOTO (Zeilennummer) weiterzuarbeiten. Einige Punkte sind jedoch zu beachten:

- Stringvariable, denen in einer Programmzeile ein String zugeordnet wird (z.B. A$="ABCDE"), müssen am Programmanfang stehen
- der STOP-Befehl und die letzte Programmzeile dürfen nicht verändert werden
- es darf keine Zeile angehängt werden (Zeilennummer > Nummer der letzten Zeile)

BEDIENUNGSHINWEISE

Das Maschinensprache-PGM wird mittels dem Basic-Loader (s. Bild 4.4.2) je nach Wunsch am Basic-RAM-Ende oder ab beliebiger vorgegebener Adresse generiert. Der hier verwendete Basic-Loader erfüllt außer der M-PGM-Generierung noch einen weiteren Zweck, was im Kapitel 5.2 "Ein spezieller Basic-Loader" eingehend erläutert wird. Der Basic-Loader zeigt die Adresse an, in die per SYS-Befehl gesprungen werden muß, wenn BREAK/CONT aktiviert werden soll.

Das folgende kurze Beispiel (s. Listing in Bild 4.4.1) soll im kurzen die Vorteile von BREAK/CONT verdeutlichen. Dieses Demo-PGM wird eingegeben und mit RUN gestartet, nachdem BREAK/CONT aktiviert worden ist.

```
10 INPUT A : STOP
20 PRINT A : STOP
30 INPUT B : STOP
40 PRINT B : STOP
50 PRINT A+B
60 END
```

Bild 4.4.1

Der INPUT-Befehl in Zeile 10 erwartet von Ihnen die Eingabe einer Zahl (mit anschließendem Return). Aufgrund des STOP-Befehls erhält man unmittelbar danach die Systemmeldung:

```
BREAK IN 10
READY.
```

LISTen Sie nun die Zeile 50 per LIST 50 aus und verändern Sie die PRINT-Anweisung zu:

```
50 PRINT A+B*B
```

und lassen Sie das (veränderte !!) PGM fortsetzen per CONT (Return). In einem solchen Fall hätten Sie, wie anfangs erklärt, sonst immer die Systemmeldung "CAN'T CONTINUE" erhalten. Jetzt jedoch wird das PGM tatsächlich in Zeile 20 fortgeführt, und die dortige PRINT-Anweisung schreibt den Wert auf den Bildschirm, den Sie vorher eingegeben haben.

Sie können in dieser Weise fortfahren und nach jedem STOP aus dem PGM herausgehen, um nach Veranderungen per CONT oder GOTO (Zeilennummer) **ohne** Datenverlust sich weiter durch das PGM durchzuarbeiten.

Während die beim COMMODORE-TOOLKIT bekannten Funktionen TRACE und STEP nur die passive Beobachterrolle ermöglichen, liefert BREAK/-CONT ein nützlich einsetzbares aktives Hilfsmittel für die PGM-Editierung und ist somit eine gut einsetzbare Betriebssystem-Hilfsroutine.

PROGRAMM-EINZELHEITEN

1.) Name: BREAK/CONT

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Betriebssystem-Zusatzroutine

4.) Anzahl der Bytes
des Basic-Loaders: 1739
des M-PGMs: 207

5a) Benutzte Variablen im Basic-Loader:
siehe Kap. 5.2 "Ein spezieller Basic-Loader"

5b) Benutzte Adressen im M-PGM:

$00,01	Zwischenspeicher für Feldende-Vektor
$02,03	Zwischenspeicher für Variablenanfang-Vektor
$04	Zwischenspeicher für die Große der erfolgten PGM-Änderungen
$3D,3E	Startadresse für CONTINUE
$FB,FC	Zwischenspeicher für Feldbeginn-Vektor
$FD,FE	Zwischenspeicher für CONTINUE-Startadresse
$FF	Art der PGM-Änderung: =$00, wenn PGM großer geworden ist =$FF, wenn PGM kleiner geworden ist

6.) Listings
des Basic-Loaders: s. Bild 4.4.2
des M-PGMs: s. Bild 4.4.3

7.) Erläuterungen zum M-PGM:

Teil 1: Zeilen 2000 - 200A
Andert den Vektor "Basic-Warmstart". Dieser Vektor enthält nach Durchlaufen dieses Teils die Adresse vom Teil 2 des M-PGMs.

```
10 REM BREAK/CONT
20 REM FKT. "CONTINUE" IST HIERMIT NACH PGM-VERAENDERUNG MOEGLICH
30 L=207
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>24083THENPRINT:PRINT"PROGRAMM IST FEHLER- HAFT EINGEGEBEN !!!":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(146):PRINT
280 PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
1000 DATA169,L14,141,2,3,169,H,141,3,3,96,108,2,3,32,96
1010 DATA197,134,122,132,123,32,115,0,170,240,240,162,255,144,6,32
1020 DATA121,197,76,225,199,72,165,45,133,2,165,46,133,3,165,49
1030 DATA133,45,133,0,165,50,133,46,133,1,165,47,133,251,165,48
1040 DATA133,252,165,61,133,253,165,62,133,254,169,L88,141,2,3,169
1050 DATAH,141,3,3,104,76,156,196,160,0,132,255,56,165,45,229
1060 DATA0,133,4,16,4,169,255,133,255,165,45,133,49,165,46,133
1070 DATA50,24,165,2,101,4,133,45,165,3,101,255,133,46,24,165
1080 DATA251,101,4,133,47,165,252,101,255,133,48,56,160,0,177,253
1090 DATA201,0,240,38,201,58,240,34,24,165,253,101,4,133,253,165
1100 DATA254,101,255,133,254,165,253,133,61,165,254,133,62,169,L14,141
1110 DATA2,3,169,H,141,3,3,76,128,196,56,165,253,233,1,133
1120 DATA0,165,254,233,0,133,1,177,0,201,144,208,203,240,214
```

Bild 4.4.2

```
2000 LDA #$0E
2002 STA $0302
2005 LDA #$20          Teil 1
2007 STA $0303
200A RTS
-------------------------------
200B JMP ($0302)
200E JSR $C560
2011 STX $7A
2013 STY $7B
2015 JSR $0073
2018 TAX               Teil 2
2019 BEQ $200B
201B LDX #$FF
201D BCC $2025
201F JSR $C579
2022 JMP $C7E1
-------------------------------
2025 PHA
2026 LDA $2D
2028 STA $02
202A LDA $2E
202C STA $03
202E LDA $31
2030 STA $2D
2032 STA $00
2034 LDA $32
2036 STA $2E           Teil 3
2038 STA $01
203A LDA $2F
203C STA $FB
203E LDA $30
2040 STA $FC
2042 LDA $3D
2044 STA $FD
2046 LDA $3E
2048 STA $FE
204A LDA #$58
204C STA $0302
204F LDA #$20
2051 STA $0303
2054 PLA
2055 JMP $C49C
-------------------------------
2058 LDY #$00
205A STY $FF
205C SEC
205D LDA $2D
205F SBC $00           Teil 4
2061 STA $04
2063 BPL $2069
2065 LDA #$FF
2067 STA $FF
2069 LDA $2D
```

```
206B STA $31
206D LDA $2E
206F STA $32
2071 CLC
2072 LDA $02
2074 ADC $04
2076 STA $2D
2078 LDA $03
207A ADC $FF
207C STA $2E
207E CLC                    Forts.
207F LDA $FB
2081 ADC $04                Teil 4
2083 STA $2F
2085 LDA $FC
2087 ADC $FF
2089 STA $30
208B SEC
208C LDY #$00
208E LDA ($FD),Y
2090 CMP #$00
2092 BEQ $20BA
2094 CMP #$3A
2096 BEQ $20BA
2098 CLC
2099 LDA $FD
209B ADC $04
209D STA $FD
209F LDA $FE
20A1 ADC $FF
20A3 STA $FE
20A5 LDA $FD
20A7 STA $3D
20A9 LDA $FE
20AB STA $3E
20AD LDA #$0E
20AF STA $0302
20B2 LDA #$20
20B4 STA $0303
20B7 JMP $C480
20BA SEC
20BB LDA $FD
20BD SBC #$01
20BF STA $00
20C1 LDA $FE
20C3 SBC #$00
20C5 STA $01
20C7 LDA ($00),Y
20C9 CMP #$90
20CB BNE $2098
20CD BEQ $20A5
```

Bild 4.4.3

Teil 2: 200B - 2022
Eingabe-Warteschleife; dieser PGM-Teil ist aus dem VC-20-Betriebssystem kopiert, es fehlt nur der Teil, der die Adresse für CONT überschreibt und damit CONT unmöglich machen würde. Der Sprung in die Betriebssystem-Routine "Einfügen und Löschen von Programmzeilen" geschieht über Teil 3.

Teil 3: 2025 - 2055
Retten der Adresse für CONT,
Retten der Vektoren auf:
- Beginn der Variablen (Basic-PGM-Ende)
- Beginn der Felder
- Ende der Felder,

Vektor "Ende der Felder" nach $2D,2E (Anfang der Variablen),
Ändern des Vektors "Basic-Warmstart", so daß er auf den Anfang des Teiles 4 zeigt,
Sprung in die Betriebssystem-Routine "Einfügen und Löschen von Programmzeilen" ($C49C)

Teil 4: 2058 - 20CD
Nach Rückkehr aus der Betriebssystem-Routine "Einfügen und Löschen von Programmzeilen" enthalten die Speicherstellen $2D und $2E die neue Adresse vom Ende der Felder. Diese Adresse wird dem Vektor "Ende der Felder" ($31,32) übergeben.
Aus den Vektoren "Ende der Felder NEU" und "Ende der Felder ALT" wird die Größe der Programmänderung errechnet und in Adresse $04 abgelegt. Wurde das Basic-PGM kleiner, dann ist dieser Wert negativ (Bit 7 = Vorzeichen) und in Adresse $FF wird #$FF eingeschrieben. Ist dieser Wert positiv, dann enthält die Adresse $FF den Wert 0.
Die geretteten Vektoren in den Adressen $00 bis 03 und $FB,FC werden durch Addition des Inhalts von $04 aktualisiert und zu den Adressen für die Zeiger auf:
- Beginn der Variablen
- Beginn der Felder

gebracht.

Die Adresse für CONT zeigt auf #$00 (Zeilenende) oder #$3A (Doppelpunkt). Ist das nicht der Fall, geschah die Programmänderung vor dem STOP-Befehl und auch dieser Vektor wird aktualisiert und zurückgebracht.

4.5 VC-OLD: NEW RÜCKGÄNGIG MACHEN

Ist es Ihnen auch schon einmal passiert, daß Sie nach einer Kaffeepause zu Ihrem Rechner zurückgekehrt sind und fast schon reflexiv, gewissermaßen aus dem Gefühl der Gründlichkeit heraus, NEW eingaben? Sicherlich schon. Meistens waren dann nur noch Basic-Leichen im Rechner. Wie ist es Ihnen aber ergangen, wenn Sie sich dann nach der NEW-Eingabe erinnerten, daß Sie vorher ein Programm eingegeben hatten, welches Sie noch nicht auf Kassette oder Diskette gespeichert hatten?

Es wird nachfolgend ein Programm vorgestellt, welches die Funktion OLD (also die Umkehrung von NEW) realisiert. Naturlich ist das keine Hexerei. Es wird hierbei nur die Tatsache ausgenutzt, daß das Betriebssystem des VC-20 bei NEW das Basic-Programm in keinster Weise anruhrt, sondern nur den Variablen- beziehungsweise Feld-Anfangs-Vektor zurücksetzt. Mit Hilfe von VC-OLD werden diese Vektoren einfach wieder auf ihren ursprünglichen Wert gesetzt, so daß das - nach wie vor vorhandene - Programm wieder benutzt, also auch geLISTet und abgeSAVEd werden kann.

BEDIENUNGSHINWEISE

VC-OLD ist ein kleines M-PGM und ist von sich aus frei verschiebbar (relocatable) und nicht nur über den speziellen Basic-Loader frei generierbar gemaß Benutzeranforderung. Es laßt sich uber unseren speziellen Basic-Loader in die Pallette von VC-20-Betriebssystem-Zusatzroutinen am Basic-RAM-Ende (Eingabe E (Return) nach PGM-Start; weitere Hinweise hierzu im Kapitel 5.2 "Ein spezieller Basic-Loader") einreihen.

Fur den Fall, daß aber Ihr Zusatzbetriebssystem gerade nicht geladen ist - man sollte es nicht für moglich halten, aber die ungunstigsten Fehlbedienungen geschehen meistens dann, wenn man nicht darauf vorbereitet ist - ist es ratsam, VC-OLD derart abzuspeichern, daß es beim Laden im Kassettenpuffer generiert wird und dieses somit das Basic-PGM nicht verletzt. Dies geschieht einfach folgendermaßen:

1.) Der Basic-Loader zu VC-OLD (siehe Bild 4.5.1) wird gestartet. Die Frage nach der Anfangsadresse wird durch Eingabe von 830 (Return) beantwortet. Die Adresse #830 gehört zum Kassettenpuffer (weiteres hierzu siehe Teil "Laden und Abspeichern"), es könnte genauso gut eine andere genommen werden, nur darf das M-PGM nicht aus dem Kassettenpuffer herausragen. 830 laßt sich eventuell besser bearbeiten als eine krumme Zahl. Das M-PGM VC-OLD steht nach dieser Eingabe im Bereich #830-#893.

2.) Datassette bzw. Floppy zum SAVEn vorbereiten.

3.) Im Direkt-Modus wird nun eingegeben:
FOR I=0 TO 63 : P$=P$+CHR$(PEEK(830+I)) : NEXT :
SAVE P$ (Return). Für Floppys muß der letzte Befehl sein: SAVE P$,8.
Hiermit wird erreicht, daß lediglich ein PGM-Name ohne PGM (der Basic-Loader von VC-OLD enthalt ja den NEW-Befehl in Zeile 240) auf Band bzw. auf Floppy abgeSAVEd wird.
Storen Sie sich nicht an den seltsamen Zeichen, die Sie beim SAVEn auf dem Bildschirm sehen werden. Diese kommen dadurch zustande, daß der VC-20 versucht, den "PGM-Namen", nämlich P$ anzuzeigen. P$ ist aber kein Name, sondern hinter P$ verbirgt sich das ganze M-PGM VC-OLD. Nach Beendigung des Abspeichervorgangs drücken Sie die Tasten (Stop) + (Restore), damit der Cursor seine Anfangsfarbe zurückerhält.

Wenn Ihnen nun jetzt in Zukunft das "NEW-Malheur" unvorbereitet passiert, so laden Sie einfach das soeben abgespeicherte "Programm" (es besteht ja nur aus einem Header) mit LOAD, drücken danach ebenso wie nach dem SAVE-Vorgang die Tasten STOP + RESTORE und geben ein: SYS 833 (Return).

Probieren Sie es mal selbst aus. Diese kleine Utility kann eine sehr wertvolle Hilfe sein, wenn Sie einmal ein PGM ausprobieren wollen, es vorher aber nicht abgespeichert haben. Wenn dann ein Programm-Absturz erfolgt, indem nichts mehr geht, noch nicht einmal irgendeine Eingabe über die Tastatur, dann bleibt normalerweise nichts anderes übrig als den VC-20 abzuschalten. Das mühsam eingegebene PGM ist dann aber auch hin. Nun, mit Hilfe von VC-OLD

gibt es einen eleganteren Ausweg aus dieser Misere: Sie drücken auf die Reset-Taste (falls vorhanden) oder bewerkstelligen RESET auf andere Weise (mehr dazu in der Begriffserklärung für "RESET" im Anhang A.2), laden VC-OLD und rufen dies, wie oben beschrieben auf.

```
10 REM VC-OLD
20 REM NEW RUECKGAENIG MACHEN
30 L=64
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$:A=VAL(A$)
210 CH=CH+A:POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>8008THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(146):PRINT:PRINT:PRINT
280 PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
1000 DATA160,3,200,177,43,208,251,200,152,24,101,43,160,0,145,43
1010 DATA133,163,152,101,44,200,145,43,133,164,136,177,163,170,200,17
1020 DATA163,240,8,177,163,133,164,134,163,144,239,200,152,101,163,133
1030 DATA45,133,47,133,49,169,0,101,164,133,46,133,48,133,50,96
```

Bild 4.5.1

Noch schneller spielt sich natürlich das eben Beschriebene ab, wenn VC-OLD sich im Bereich Ihrer Betriebssystem-Zusatzroutinen am Basic-RAM-Ende befindet. Nach NEW bzw. RESET rufen Sie VC-OLD dann einfach mit SYS Anfangsadresse (Return) auf, wobei die Anfangsadresse ja vom Basic-Loader angegeben wird und diese von Ihnen notiert wird.

Die wahrscheinlich eleganteste Lösung wird wohl sein, wenn Sie anstatt von SYS Anfangsadresse (Return) einfach nur OLD (Return) eingeben, was bei Verwendung des PGMs VC-COMMAND (siehe Kapitel 4.1) realisiert werden kann.

PROGRAMM-EINZELHEITEN

1.) Name: VC-OLD

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Betriebssystem-Zusatzroutine

4.) Anzahl der Bytes
des Basic-Loaders: 1025
des M-PGMs: 64

5.) Benutzte Adressen:

$A3	LOW-Link zur nachsten Basic-Zeile
$A4	HIGH-Link " " " "

6.) Listings
des Basic-Loaders: siehe Bild 4.5.1
des M-PGMs: siehe Bild 4.5.2

7.) Erlauterungen zum M-PGM:

Teil 1: Zeilen 2000 - 2005
Es wird das Ende der ersten Basic-Zeile gesucht, also die erste Null, welche nach den ersten vier Bytes des Basic-Anfangs folgt (mehr hierzu siehe Kap. 2.1 "Speicherorganisation bei einem Basic-Programm").

Teil 2: Zeilen 2007 - 2018
Das Register Y enthalt den Offset zwischen dem Ende der 1. Basic-PGM-Zeile und dem Basic-Anfang. In diesem Teil wird der Link zur 2. Basic-PGM-Zeile bzw. zum Ende des Basic-PGMs (falls es nur aus einer Zeile besteht) einfach durch Addition vom Offset Y zur Basic-Anfangsadresse errechnet. Dieser Wert wird in die ersten 2 Bytes der ersten Basic-PGM-Zeile und zwecks weiterer Verarbeitung in die Adressen $A3 und $A4 hineingeschrieben.

Teil 3: Zeilen 201A - 2029
Das Adreßpaar $A3,$A4 zeigt auf das 1. Byte des Link-Paares, welches in der Basic-Folgezeile am Anfang steht, jedoch sei-

nerseits auf die darauffolgende Folgezeile zeigt.
Der Mechanismus in Teil 3 tut also nichts weiter als von Link zu Link springen, bis ein Link-Paar aus 2 Nullen besteht.

Teil 4: Zeilen 202B - 203F
2 Nullen als Link-Paar kennzeichnen das Basic-PGM-Ende. Es verbleibt nur noch die Aufgabe, die erkannte Basic-PGM-Ende-adresse um 2 zu erhöhen und diesen Wert (= 2 Bytes) in die Speicherzellen für die Vektoren: Variablenanfang ($2D,$2E), Feldanfang ($2F,$30) sowie Feldende ($31,$32) hineinzuschreiben.

```
2000 LDY #$03
2002 INY                  Offset Y auf BASIC-Anfang + 4  stellen
2003 LDA ($2B),Y
2005 BNE $2002            Spr. w. Ende der 1. Zeile nicht gefunden
2007 INY
2008 TYA                  Y=Offset zur 2. Zeile
2009 CLC
200A ADC $2B              ($2B),0 , ($2C),1 = $2B,$2C +Y
200C LDY #$00
200E STA ($2B),Y              und $A3,$A4 = $2B,$2C +Y
2010 STA $A3
2012 TYA
2013 ADC $2C
2015 INY
2016 STA ($2B),Y
2018 STA $A4
201A DEY                  Y=0
201B LDA ($A3),Y
201D TAX                  X=LOW-Link
201E INY                  Y=1
201F ORA ($A3),Y
2021 BEQ $202B            Sprung, wenn Y-1=Offset fuer BASIC-Ende
2023 LDA ($A3),Y          A=HIGH-Link
2025 STA $A4              $A3,$A4 = X,A
2027 STX $A3
2029 BCC $201A
202B INY                  Y=2
202C TYA
202D ADC $A3              Variablenanfangvektor:
202F STA $2D                  $2D,$2E = $A3,$A4 +Y+2
2031 STA $2F
2033 STA $31                  Feldanfangvektor = Feldendevektor =
2035 LDA #$00                         Variablenanfangvektor
2037 ADC $A4
2039 STA $2E
203B STA $30
203D STA $32
203F RTS
```

Bild 4.5.2

4.6 VC-CHECKSUM

Stellen Sie sich nicht auch ab und zu die Frage: "Habe ich wirklich das gleiche Programm geladen, welches ich vor 14 Tagen abgeSAVEd hatte ?" oder "Ist mein Programm auch nach einem Programmlauf das gleiche wie zuvor ?" Letztere Frage ist keinesfalls abwegig. Denken wir nur an Programme, die voll gespickt sind mit PEEK- und POKE-Befehlen. Wie schnell kann es passieren, daß man dabei aus Versehen ins eigene Programm hineinPOKEd.

Die eigentlich Methode, seine Daten bzw. Programme per Checksummenprufung auf Fehlerfreiheit zu überprüfen, wird von den meisten als praktisch und einfach akzeptiert. VC-CHECKSUM führt diese Berechnung durch und läßt sich in zweierlei Hinsicht benutzen:

- Checksummen-Berechnung für Basic-Programme
- Checksummen-Berechnung für beliebige Speicherbereiche

Die Checksummen-Berechnung besteht darin, daß die Werte aller Bytes des Basic-Programms bzw. des angegebenen Speicherbereichs addiert werden.

VC-CHECKSUM wird je nach Wunsch in einem beliebigen Speicherbereich generiert. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen Überschreiben geschützt und sie können es immer dann aufrufen, wenn Sie eine Checksummenberechnung benötigen. Weiterhin ist es koppelbar mit von Ihnen frei definierbaren Befehlen (s. Kap. 4.1).

BEDIENUNGSHINWEISE

VC-CHECKSUM ist ein Maschinen-Programm (s. Bild 4.6.2), das in den im Kapitel 5.2 beschriebenen Basic-Loader (s. Bild 4.6.1) eingebunden wurde. Detailinformation bezgl. diesem kann dort in Erfahrung gebracht werden. Der Basic-Loader wird normal mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das M-PGM generiert werden soll. Bei Eingabe von "E" wird es ans Basic-RAM-Ende gespeichert, wobei eventuell schon andere vorhandene M-PGMe in keinster Weise angetastet werden.

4.6 CHECKSUM

```
10 REM VC-CHECKSUM
20 REM CHECKSUMMENBERECHNUNG VOM BASIC-PGM ODER VON ANGEGEBENEM SPEICHERBEREICH
30 L=134
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>17205THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(157)",B"CHR$(146):PRINT"FUER BASIC-PGM"
280 PRINT:PRINTCHR$(18)"SYS"SDCHR$(157)",DEZ-A,DEZ-E"CHR$(146)
290 PRINT"A=ANFANGSADRESSE":PRINT"E=ENDADRESSE":PRINT
300 PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
1000 DATA32,115,0,201,66,240,31,32,L69,H,134,251,132,252,32,L66
1010 DATAH,134,253,132,254,169,13,32,122,242,32,L80,H,32,205,221
1020 DATA169,13,32,122,242,96,32,115,0,165,43,133,251,165,44,133
1030 DATA252,165,46,133,254,165,45,133,253,208,2,198,254,198,253,76
1040 DATAL21,H,32,115,0,32,138,205,32,247,215,166,20,164,21,96
1050 DATA169,0,141,179,3,141,180,3,168,165,252,197,254,144,8,208
1060 DATA30,165,253,197,251,144,24,177,251,24,109,179,3,141,179,3
1070 DATA152,109,180,3,141,180,3,230,251,208,222,230,252,208,218,174
1080 DATA179,3,173,180,3,96
```

Bild 4.6.1

Die Eingabe "E" wurde die Anzeige:

```
SYS 16243,B
SYS 16243,AE,EA
```

auf dem Bildschirm zur Folge haben, wenn der VC-20 mit einer 8K-Byte-Speichererweiterung versehen ist. Wie werden diese beiden SYS-Befehle nun angewandt:

Nehmen wir an, Sie haben ein Basic-Programm im Arbeitsspeicher und wollen eine CHECKSUM-Berechnung durchführen. Sie brauchen nur eingeben:

```
SYS 16243,B  (Return)
```

und erhalten daraufhin eine Zahl, die Sie sich irgendwohin notieren, falls Sie später einmal einen Vergleich ziehen wollen. Wenn Sie übrigens den Basic-Loader in Bild 4.6.1 der Funktion CHECKSUM unterziehen, nachdem Sie in Zeile 240 das NEW durch END ersetzt haben, so erhalten Sie den Wert 41777.

Der CHECKSUM-Wert für Basic-Programme ist nicht unabhängig von der Lage des Basic-Programms. Dies ist einfach zu verstehen, wenn man das Speicherungsprinzip bei einem Basic-Programm kennt (s. Kap. 2.1). Ein Basic-Programm im Arbeitsspeicher besteht ja keineswegs identisch aus den bei LIST angezeigten Zeichen, sondern ist durchsetzt von absoluten Adressen, die das VC-20-Betriebssystem zur Auseinanderhaltung der Zeilen benötigt. Deswegen wäre also der CHECKSUM-Wert schon bereits dann ein anderer, wenn Sie das Basic-Programm um nur 1 Byte verschieben. Falls Sie den CHECKSUM-Basic-Loader mit der o.g. Veränderung der Zeile 240 in der Grundversion gespeichert haben, so erhalten Sie anstatt des Wertes 41777, den Wert 41699.

Die zweite Benutzungsregel von CHECKSUM funktioniert in der Weise, daß Sie anstatt des B für die Basic-Programm-Überprüfung die dezimale Anfangsadresse, gefolgt von einem Komma und der dezimalen Endadresse (incl.) , eines von Ihnen gewünschten zu überprüfenden Speicherbereichs eingeben. Nehmen wir an, dieser Bereich liegt zwischen $2000 und $2100, so müßten Sie eingeben:

SYS 16243,8192,8447 (Return)

Die Adressenwerte können auch durch Variable oder beliebige Terme ersetzt werden.

PROGRAMM-EINZELHEITEN

1.) Name: VC-CHECKSUM

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Basic-Loader

```
2000 JSR $0073          Hole naechstes Zeichen (nach Komma)
2003 CMP #$42
2005 BEQ $2026          Sprung, wenn "B"
2007 JSR $2045          1. DEZ-Term nach $FB,$FC
200A STA $FB
200C STY $FC
200E JSR $2042          2. DEZ-Term nach $FD,$FE
2011 STA $FD
2013 STY $FE
2015 LDA #$0D           CR ausgeben
2017 JSR $F27A
201A JSR $2050          Checksumme berechnen
201D JSR $BDCD          Checksumme ausgeben
2020 LDA #$0D           CR ausgeben
2022 JSR $F27A
2025 RTS
-------------------------------------------------------
2026 JSR $0073          zum Eingabeende gehen
2029 LDA $2B            BASIC-Anfang-Vektor nach $FB,$FC
202B STA $FB
202D LDA $2C
202F STA $FC
2031 LDA $2E            Variablenanfang-Vektor = BASIC-Ende-
2033 STA $FE                Vektor-1 nach $FD,$FE
2035 LDA $2D
2037 STA $FD
2039 BNE $203D          Vermindere Variablenanfang-Vektor
203B DEC $FE
203D DEC $FD
203F JMP $2015
-------------------------------------------------------
2042 JSR $0073          Hole naechstes Zeichen
2045 JSR $CD8A          DEZ-Term auswerten
2048 JSR $D7F7          Umwandl. in Integerzahl und nach $14,$15
204B LDX $14
204D LDY $15            X,Y = L,H von DEZ-Term
204F RTS
-------------------------------------------------------
2050 LDA #$00           Initialisierung Checksummenberechnung
2052 STA $03B3
2055 STA $03B4
2058 TAY
2059 LDA $FC
205B CMP $FE
205D BCC $2067          Sprung, wenn ($FC)<($FD)
205F BNE $207F          Sprung, wenn ($FE)<>($FC)
2061 LDA $FD
2063 CMP $FB
2065 BCC $207F          Sprung, wenn ($FD)<($FB)
2067 LDA ($FB),Y        Byte aus Bereich nach A
2069 CLC
206A ADC $03B3          Checksumme = Checksumme + Byte
206D STA $03B3
2070 TYA
2071 ADC $03B4
2074 STA $03B4
2077 INC $FB            naechstes Byte adressieren ($FB,$FC)
2079 BNE $2059
207B INC $FC
207D BNE $2059          Sprung: immer
-------------------------------------------------------
207F LDX $03B3          Checksumme aus ($03B3,$03B4) holen
2082 LDA $03B4              und nach X,A
2085 RTS
```

Bild 4.6.2

4.) Anzahl der Bytes
des Basic-Loaders: 1610
des M-PGMs: 134

5a) Benutzte Variablen im Basic-Loader: s. Kap. 5.2

5b) Benutzte Adressen im M-PGM:

$FB,FC	Speicher für 1. DEZ-Wert = Vektor auf Speicherbereichsanfang
$FD,FE	Speicher für 2. DEZ-Wert = Vektor auf Speichererbereichsende
$03B3,03B4	Augenblicklicher Checksummenwert

6.) Listings
des Basic-Loaders: Bild 4.6.1
des M-PGMs: Bild 4.6.2

7.) Erläuterungen zum Basic-Loader werden in Kap. 5.2 gegeben und zum M-PGM implizit im gut kommentierten Listing (Bild 4.6.2).

4.7 REM-INVERTIERER

Sie werden sich vielleicht sagen: "REM-Invertierer? Habe ich schon irgendwo gelesen. Ist nichts Neues." Im Prinzip haben Sie da sicherlich recht. Nur, haben Sie schon einen REM-Invertierer kennengelernt, bei dem

- keine Einschrankung bei der Gestaltung der REM-Texte besteht ? Wir haben zum Beispiel schon Beschreibungen in den Händen gehabt, die besagten, daß Sie nach dem Eingeben von REM zwei Blanks schreiben mussen, etc.

- keine Verkurzung der REM-Zeilen vorgenommen wird ? Mit anderen Worten, das Programm ist nach Benutzung der REM-Invertierer-Funktion genauso gestaltet wie vorher, nur erscheinen die Texte in den REM-Zeilen in Revers-Darstellung.

Der hier vorgestellte REM-Invertierer bietet diese Vorzüge. Es ist ein Maschinenprogramm (s. Bild 4.7.2), welches je nach Wunsch in einem beliebigen Speicherbereich generiert werden kann. Wenn

Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen Überschreiben geschützt und sie können es auch dann aufrufen, wenn Sie beliebige andere Programme im Arbeitsspeicher haben. Weiterhin ist es koppelbar mit von Ihnen frei definierbaren Befehlen (s. Kap. 4.1).

BEDIENUNGSHINWEISE

```
10 REM REM-INVERTIERER
30 L=169
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$)"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>24296THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(146):PRINT
280 PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
1000 DATA165,43,133,163,165,44,133,164,160,0,177,163,208,6,200,177
1010 DATA163,208,1,96,160,3,200,177,163,208,15,200,152,24,101,163
1020 DATA133,163,169,0,101,164,133,164,208,222,201,143,208,232,200,177
1030 DATA163,240,232,201,32,240,247,169,18,209,163,240,222,170,165,163
1040 DATA133,187,165,164,133,188,177,187,72,138,145,187,230,187,208,2
1050 DATA230,188,104,170,208,240,177,187,208,236,200,177,187,240,3,136
1060 DATA208,228,200,145,187,200,152,24,101,187,133,187,169,0,101,188
1070 DATA133,46,133,48,133,50,165,187,133,45,133,47,133,49,160,1
1080 DATA177,163,208,3,76,L0,H,136,177,163,170,232,138,145,163,141
1090 DATA60,3,200,138,208,7,177,163,170,232,138,145,163,177,163,133
1100 DATA164,173,60,3,133,163,76,L128,H
```

Bild 4.7.1

Der REM-Invertierer ist ein Maschinen-Programm, das in den im Kapitel 5.2 beschriebenen Basic-Loader (s. Bild 4.7.1) eingebunden

wurde. Detailinformation bezgl. diesem kann dort in Erfahrung gebracht werden. Der Basic-Loader wird normal mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das M-PGM generiert werden soll. Bei Eingabe von "E" wird es ans Basic-RAM-Ende gespeichert, wobei eventuell schon andere vorhandene M-PGMe in keinster Weise angetastet werden.

Wenn Sie beispielsweise einen VC-20 in der Grundversion besitzen und sich für das Ablegen des M-PGMs am Basic-RAM-Ende entscheiden, so erhalten Sie auf dem Bildschirm die Meldung:

```
M-PGM-AUFRUF MIT:
SYS 7504
```

Laden Sie mal eines Ihrer Programme, das recht großzügig mit REM-Zeilen gespickt ist und geben Sie SYS 7504 ein. Der Invertierungsprozeß dauert nur Sekundenbruchteile. Danach tippen Sie LIST und werden hernach das gewünschte Resultat betrachten können. Auch können Sie das neu geartete Programm normal drucken lassen oder abSAVEn.

Wenn Sie die eine oder andere REM-Zeile wieder in den ursprünglichen Zustand zuruckversetzen wollen, gehen Sie einfach mit dem Cursor an eine beliebige Stelle dieser Zeile, drücken RETURN und schon ist's geschehen. Beim darauffolgenden LIST können Sie sich von der Rückgängigmachung überzeugen.

PROGRAMM-EINZELHEITEN

1.) Name: REM-INVERTIERER

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Basic-Loader

4.) Anzahl der Bytes
des Basic-Loaders: 1586
des M-PGMs: 169

5a) Benutzte Variablen im Basic-Loader: s. Kap. 5.2

```
2000 LDA $2B              BASIC-Anfang-Vektor nach  $A3,$A4
2002 STA $A3
2004 LDA $2C
2006 STA $A4
2008 LDY #$00
200A LDA ($A3),Y
200C BNE $2014            Sprung, wenn LOW-Link<>0
200E INY
200F LDA ($A3),Y
2011 BNE $2014            Sprung, wenn HIGH-Link<>0
2013 RTS                  am BASIC-Ende angekommen
2014 LDY #$03             Y initialisieren
2016 INY
2017 LDA ($A3),Y          1. Zeichen in der naechsten BASIC-Zeile
2019 BNE $202A            Sprung, wenn noch nicht Zeilenende
201B INY                  Zeilenende erreicht, zeigt auf LOW-Link
201C TYA                   $A3,$A4 = $A3,$A4 + Y
201D CLC
201E ADC $A3
2020 STA $A3
2022 LDA #$00
2024 ADC $A4
2026 STA $A4
2028 BNE $2008            Sprung: immer
202A CMP #$8F
202C BNE $2016            Sprung, wenn nicht "REM"
202E INY
202F LDA ($A3),Y
2031 BEQ $201B
2033 CMP #$20
2035 BEQ $202E            Sprung, wenn " "
2037 LDA #$12
2039 CMP ($A3),Y
203B BEQ $201B            Sprung, wenn "REM" bereits invertiert
203D TAX                  X="RVS ON"
203E LDA $A3               $BB,$BC = $A3,$A4
2040 STA $BB
2042 LDA $A4
2044 STA $BC
2046 LDA ($BB),Y          BASIC-Rest um 1 Stelle
2048 PHA                      nach oben schieben
2049 TXA
204A STA ($BB),Y
204C INC $BB
204E BNE $2052
2050 INC $BC
2052 PLA
2053 TAX
2054 BNE $2046            Sprung, wenn keine "0" gefunden
```

Bild 4.7.2 (Teil 1)

```
2056 LDA ($BB),Y
2058 BNE $2046        Sprung, wenn keine 2. "0" gefunden
205A INY
205B LDA ($BB),Y
205D BEQ $2062        Sprung, wenn 3. "0" gefunden
205F DEY
2060 BNE $2046        Sprung, wenn keine 3. "0" gefunden
2062 INY
2063 STA ($BB),Y      3. "0" ablegen = BASIC-Ende
2065 INY              Y=Y+1
2066 TYA               $BB,$BC = $BB,$BC +Y
2067 CLC
2068 ADC $BB
206A STA $BB
206C LDA #$00
206E ADC $BC
2070 STA $2E          Vektoren:     Variablenanfang HIGH
2072 STA $30                        Feldanfang      HIGH
2074 STA $32              neu       Feldende        HIGH
2076 LDA $BB
2078 STA $2D              justie-   Variablenanfang LOW
207A STA $2F                        Feldanfang      LOW
207C STA $31              ren       Feldende        LOW
207E LDY #$01
2080 LDA ($A3),Y
2082 BNE $2087        Sprung, wenn nicht BASIC-Ende
2084 JMP $2000
2087 DEY
2088 LDA ($A3),Y      naechsten Link um 1 vergroessern
208A TAX
208B INX
208C TXA
208D STA ($A3),Y
208F STA $033C        LOW-Link zwischenspeichern
2092 INY
2093 TXA
2094 BNE $209D
2096 LDA ($A3),Y
2098 TAX
2099 INX
209A TXA
209B STA ($A3),Y
209D LDA ($A3),Y       $A3,$A4 = ($A3),0 , ($A3),1
209F STA $A4
20A1 LDA $033C
20A4 STA $A3
20A6 JMP $2080        zur Pruefung auf BASIC-Ende
```

Bild 4.7.2 (Teil 2)

5b) Benutzte Adressen im M-PGM:

$A3,A4	Zeilen-Anfangsadresse
$BB,BC	Zwischenspeicher für $A3,A4

6.) Listings

des Basic-Loaders:	Bild 4.7.1
des M-PGMs:	Bild 4.7.2 (Teil 1 u. Teil 2)

7.) Erläuterungen zum Basic-Loader werden in Kap. 5.2 gegeben und zum M-PGM implizit im gut kommentierten Listing (Bild 4.7.2).

4.8 RENUMBER

Bisher wurde schon eine ganze Reihe von RENUMBER-Programmen veröffentlicht. Unseres Wissens nach konnten diese zwar ordentlich die Zeilennummern am Anfang der Zeile neu durchnumerieren, jedoch keines von denen auch die in den Zeilen vorkommenden Zeilennummern. Betroffen sind die Befehle GOTO, GOSUB, THEN, ON, LIST und RUN, hinter denen ja (nicht notwendigerweise bei THEN, LIST, RUN) Zeilennummern stehen, die im Programm vorkommen.

VC-RENUMBER (s. Bild 4.8.1) rechnet schnell und zuverlässig auch die im Programm auftauchenden Zeilennummern um. Stößt es zufällig einmal auf eine Zeilennummer, die als Programmzeile nicht vorkommt (Nobody is perfect beim Programmieren), so wird an diese Stelle beim Umnumererierungsverfahren die Zeilennummer 63999 (also die höchstmögliche) gesetzt.

Das Maschinenprogramm wird je nach Ihrem Wunsch in einem beliebigen Speicherbereich generiert. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen Überschreiben geschützt und sie können es beliebig oft aufrufen, wenn Sie ein im Arbeitsspeicher vorhandenes Basic-Programm umnumerieren lassen wollen. Weiterhin ist es koppelbar mit von Ihnen frei definierbaren Befehlen (s. Kap. 4.1).

BEDIENUNGSHINWEISE

VC-RENUMBER ist ein Maschinen-Programm, das in den im Kapitel 5.2 beschriebenen Basic-Loader (s. Bild 4.8.1) eingebunden wurde.

```
10 REM VC-RENUMBER
20 REM AENDERUNG DER ZEILENNUMERIERUNG
30 L=589
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$)"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>76200THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(157)",(ZN),(DF)"CHR$(146):PRINT
275 PRINT"ZN=NUMMER DER 1.ZEILE DF=ZEILENNR.-ABSTAND"
276 PRINT:PRINT",(DF) BZW. ,(ZN),(DF) KOENNEN ENTFALLEN":PRINT
280 PRINT:PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
1000 DATA32,L457,H,32,L480,H,165,43,133,95,165,44,133,96,32,L559
1010 DATAH,240,8,32,L539,H,144,246,76,L451,H,32,L574,H,32,L559
1020 DATAH,208,43,165,167,133,164,165,168,133,165,165,43,133,95,165
1030 DATA44,133,96,160,3,165,165,145,95,136,165,164,145,95,32,L539
1040 DATAH,32,L559,H,208,237,169,255,133,58,76,116,196,96,160,4
1050 DATA132,15,177,95,240,200,201,34,208,8,165,15,73,255,133,15
1060 DATA208,18,36,15,48,14,201,143,240,180,162,6,221,L582,H,240
1070 DATA6,202,208,248,200,208,219,24,152,101,95,133,122,133,90,166
1080 DATA96,144,1,232,134,123,134,91,32,115,0,144,10,201,171,240
1090 DATA57,201,164,208,223,240,51,32,107,201,32,L226,H,165,90,166
1100 DATA91,133,122,134,123,162,0,160,0,189,1,1,240,15,72,32
1110 DATA115,0,144,3,32,L300,H,104,145,122,232,208,236,32,115,0
1120 DATA32,121,0,176,5,32,L409,H,240,246,170,56,165,122,229,95
1130 DATA168,138,201,44,240,161,201,171,240,157,201,164,240,153,170,76
1140 DATAL84,H,165,167,166,168,133,164,134,165,165,43,166,44,133,36
1150 DATA134,37,160,3,177,36,197,21,208,24,136,177,36,197,20,208
1160 DATA17,165,165,166,164,133,98,134,99,162,144,56,32,73,220,76
1170 DATA221,221,32,L539,H,160,1,177,36,208,6,169,249,162,255,208
1180 DATA228,170,136,177,36,134,37,133,36,76,L242,H,134,169,166,45
1190 DATA164,46,134,88,132,89,232,208,1,200,228,55,152,229,56,144
1200 DATA3,76,53,196,132,46,134,45,160,1,162,0,161,88,145,88
1210 DATA165,88,208,2,198,89,198,88,165,88,197,122,165,89,229,123
1220 DATA176,234,8,165,95,166,96,133,90,134,91,40,160,1,177,90
1230 DATA208,4,166,169,136,96,170,136,177,90,168,176,6,200,208,7
1240 DATA232,208,4,208,1,202,136,152,160,0,145,90,72,138,200,145
1250 DATA90,133,91,104,133,90,76,L366,H,165,45,208,2,198,46,198
1260 DATA45,165,122,166,123,133,88,134,89,160,1,162,0,177,88,129
1270 DATA88,230,88,208,2,230,89,165,88,197,45,165,89,229,46,144
1280 DATA236,176,159,32,L457,H,76,72,210,169,10,141,226,3,169,0
1290 DATA141,227,3,133,168,133,163,133,165,169,100,133,167,133,164,96
1300 DATA32,121,0,240,53,32,115,0,176,23,32,107,201,72,165,21
```

Bild 4.8.1 (Teil 1)

```
1310 DATA166,20,133,168,134,167,133,165,134,164,104,240,29,201,44,240
1320 DATA3,76,8,207,32,115,0,176,248,32,107,201,72,165,21,166
1330 DATA20,141,227,3,142,226,3,104,208,231,96,24,165,164,109,226
1340 DATA3,133,164,165,165,109,227,3,133,165,176,2,201,250,96,160
1350 DATA0,177,95,170,200,177,95,134,95,133,96,177,95,96,169,43
1360 DATA133,95,162,0,134,96,96,155,138,167,137,141,203
```

Bild 4.8.1 (Teil 2)

Detailinformation bezgl. diesem kann dort in Erfahrung gebracht werden. Der Basic-Loader wird normal mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das M-PGM generiert werden soll. Bei Eingabe von "E" wird es ans Basic-RAM-Ende gespeichert, wobei eventuell schon andere vorhandene M-PGMe in keinster Weise angetastet werden.

Nachdem Sie sich für eine Anfangsadresse oder für "E" entschieden und nach entsprechender Eingabe RETURN gedrückt haben, wird auf dem Bildschirm der SYS-Befehl angezeigt, der die RENUMBER-Funktion ausführt.

Wenn Sie beispielsweise eine 8-KByte-Speichererweiterung besitzen und sich für das Abspeichern von VC-RENUMBER am RAM-Ende entschieden haben, so würde ausgegeben werden:

SYS 15788,(ZN),(DF) (Return)

Die Angaben ",(DF)" oder von ",(ZN),(DF)" sind optional. (ZN) steht für die gewünschte Anfangszeilennummer, (DF) für die Differenz der Zeilennummern. Die Eingabe:

SYS 15788,1000,20 (Return)

würde demnach bewirken, daß das im Arbeitsspeicher vorhandene Basic-Programm ab Zeilennummer 1000 im Abstand 20 neu durchnumeriert würde.

VC-RENUMBER kann nur ein gesamtes Programm neu durchnumerieren. Für den Programmierer kann das eventuell dann ein Nachteil sein, wenn er beabsichtigt, mittels Gebrauch bestimmter Zeilennummernbereiche Programmteile untereinander abzugrenzen. Z.B. soll also Unterprogramm 1 im Zeilennummernbereich 2000-2999 sein, Unterprogramm 2 im Bereich 3000-3999, etc. Wenn einmal Änderungen an ei-

nem solchen Programm vorgenommen werden und RENUMBER auf das geänderte Programm angewandt wird, so dürfte die ehemalige Zuordnung zu Zeilennummernbereiche wohl hin sein.

Abhilfe schafft in diesem Fall das im Kapitel 4.2 besprochene Programm VC-KEEP, mit dessen Hilfe sich beliebige Programmteile im vorhandenen RAM zwischenspeichern und auf Bedarf wieder zurückholen lassen. RENUMBER wird dann eben nur auf Programmteile angewandt, die sich über die Funktion FETCH (s. Kap 4.2) nach Belieben mit anderen Programmteilen verknüpfen lassen.

PROGRAMM-EINZELHEITEN

1.) Name: VC-RENUMBER

2.) Ausbaustufe: beliebig, allerdings mit der Einschränkung, daß für einen VC-20 in GV der Basic-Loader ein wenig kürzer gemacht werden muß.
In diesem Fall sind folgende Zeilen wegzulassen: 10, 20, 40, 50, 60, 70, 110, 120, 220, 275, 276 und 280. In Zeile 270 muß die Anweisung RETURN stehen.
Die RENUMBER-Funktion wird aufgerufen per: SYS 7084,(ZN),(DF) (Return)

3.) Art des PGMs: Basic-Loader

4.) Anzahl der Bytes
des Basic-Loaders: 3319 (2775 in der GV-Fassung)
des M-PGMs: 589

5a) Benutzte Variablen im Basic-Loader: s. Kap. 5.2

5b) Benutzte Adressen im M-PGM:
$24-25, $58-5B, $5F-60, $62-63, $A3-A9, $03E2-03E3

6.) Listing des Basic-Loaders: Bild 4.8.1

7.) Erläuterungen zum Basic-Loader werden in Kap. 5.2 gegeben.

4.9 GOTO UND GOSUB IN VERBINDUNG MIT VARIABLEN

Haben Sie sich als Basic-Programmierer nicht auch schon einmal einen Sprungbefehl gewünscht, bei dem es nicht nötig ist, eine absolute Zeilennummer anzugeben, sondern der ebenso mit einer Variablen oder sogar mit einem belieben Term seine Dienst tut ?

Hier ist er: wir haben ihn JUMP genannt. Hinter JUMP können Sie eine Variable, Feldvariable oder einen numerischen Term setzen oder auch ganz auf eine Angabe verzichten. Im einen Fall wird das Programm den Wert der Variablen abrufen bzw. den Term berechnen und zu derjenigen Zeile springen, deren Zeilennummer diesem Wert entspricht. Im anderen Fall, bei Verzicht einer Angabe, springt das Programm zum Anfang und setzt seine Arbeit in der 1. Zeile fort, ohne allerdings dabei irgendeine Variable zu loschen. Wenn man namlich den Befehl RUN irgendwo ins PGM einbaut, dann kehrt zwar das PGM auch zum Anfang zurück, CLEARt aber sämtliche Variablen.

Mit dem hier vorgefuhrten Programm JUMP/VARSUB ist es auch möglich, Unterprogrammaufrufe per Variable bzw. Term vorzunehmen. Das luxuriosere GOSUB haben wir hier als VARSUB bezeichnet.

JUMP/VARSUB ist ein Maschinenprogramm, welches je nach Wunsch in einem beliebigen Speicherbereich generiert wird. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen überschreiben ge schützt und sie konnen es immer dann aufrufen, wenn Sie es beno tigen. Weiterhin mussen die Funktionen JUMP und VARSUB mit von Ihnen frei definierbaren Befehlen (Sie können natürlich diese beiden Bezeichnungen ubernehmen) per Programm VC-COMMAND (s. Kap. 4.1) mit den vorgegebenen SYS-Adressen (s. Bedienungshinweise im Anschluß) gekoppelt werden.

BEDIENUNGSHINWEISE

JUMP/VARSUB ist ein Maschinen-Programm (s. Bild 4.9.3), das in den im Kapitel 5.2 beschriebenen Basic-Loader (s. Bild 4.9.1) eingebunden wurde. Detailinformation bezgl. diesem kann dort in Erfahrung gebracht werden. Der Basic-Loader wird normal mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das

M-PGM generiert werden soll. Bei Eingabe von "E" wird es ans Basic-RAM-Ende gespeichert, wobei eventuell schon andere vorhandene M-PGMe in keinster Weise angetastet werden.

```
10 REM JUMP/VARSUB
20 REM VARIABLES GOTO/GOSUB (NUR IN VERBINDUNG MIT PGM VC-COMMAND)
25 REM SYNTAX: "JUMP"=GEHE ZU PGM-ANFANG
26 REM "JUMP(NUM. TERM)"
27 REM "VARSUB(NUM. TERM)"
30 L=46
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):AD=AD+SD:GOSUB250:A=AL:GOTO210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO240
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(146):PRINT"=JUMP BZW.":PRINT"=JUMP(BEL.TERM)
275 PRINT:PRINTCHR$(18)"SYS"SD+8CHR$(146):PRINT"=VARSUB(BEL.TERM)
280 PRINT:PRINT:PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
1000 DATA32,121,0,208,32,76,142,198,169,3,32,251,195,165,123,72
1010 DATA165,122,72,165,58,72,165,57,72,169,141,72,32,121,0,32
1020 DATAL37,H,76,174,199,32,138,205,32,247,215,76,163,200
```

Bild 4.9.1

Als Beispiel wählen wir den Fall, daß Sie einen VC-20 ohne jegliche Speichererweiterung haben, also einen VC-20 in Grundversion (GV). Per Eingabe von "E" entscheiden Sie sich dafür, daß das M-PGM JUMP/VARSUB am Ende des Basic-RAM generiert wird. Sie erhalten sodann die Anzeige von zwei SYS-Adressen. Speziell in diesem Beispiel sind dies die Befehle:

```
SYS 7627   für die Funktion JUMP
SYS 7635   "   "   "        VARSUB
```

Eine Kopplung mit beliebigen Befehlsworten ist nicht notwendig, wenn Sie JUMP nur in der Funktion benötigen, ohne irgendeine Angabe zum Programmanfang zu springen. In diesem Fall können Sie jetzt beginnen, ein beliebiges Basic-PGM zu kreieren. Immer dort, wo Sie den Sprung zum Programmanfang wünschen, welche Zeilennummer die 1. Zeile auch immer haben mag - daß man diese nicht wissen muß, ist ja gerade der Witz hierbei -, setzen Sie den Befehl SYS 7627 ein. Bei diesem Sprung werden, wie schon am Anfang erwähnt, die Variablenwerte nicht geloscht.

Falls Sie aber doch Variablen oder numerische Terme als Sprungziele verwenden wollen, so wurden Sie per Programm VC-COMMAND aus Kapitel 4.1 die Befehle JUMP und VARSUB (andere Namen sind natürlich im Rahmen der in Kap. 4.1 besprochenen Einschränkungen auch möglich) mit den Einsprungadressen 7627 und 7635 koppeln müssen. Wenn dies geschehen ist, funktioniert z.B. das Programm in Bild 4.9.2.

```
20 A=A+1
30 JUMP A*100
100 PRINT1:JUMP
200 PRINT2:JUMP
300 PRINT3:JUMP
400 PRINT4:JUMP
500 PRINT5:JUMP
600 PRINT6:JUMP
700 PRINT7:JUMP
800 PRINT8:JUMP
900 PRINT9:JUMP
1000 PRINT10
```

Bild 4.9.2

Dies tut zwar nichts besonders Gescheites, demonstriert jedoch im Kurzen, wie JUMP eingesetzt werden kann. Nach RUN wird dieses PGM untereindander die Zahlen 1 bis 10 anzeigen.

Eine kleine Einschränkung der beiden neuen Befehle: wenn sie in einer IF-THEN-Anweisung zum Einsatz kommen sollen, so ist ein Doppelpunkt zwischen THEN und dem neuen Befehl notwendig, also z.B.:

```
30 IF B=0 THEN:JUMP A*100
```

```
2000 JSR $0079        Hole momentanes Zeichen
2003 BNE $2025        Sprung, wenn numerischer Term vorhanden
2005 JMP $C68E        Sprung zum Anfang des Programms
2008 LDA #$03         Einsprung fuer VARSUB
200A JSR $C3FB        Ist im Stack Platz?
200D LDA $7B
200F PHA
2010 LDA $7A          BASIC-Ruecksprung garantieren
2012 PHA
2013 LDA $3A
2015 PHA
2016 LDA $39
2018 PHA
2019 LDA #$8D
201B PHA
201C JSR $0079        Hole momentanes Zeichen
201F JSR $2025        BASIC-Zeilenadresse holen
2022 JMP $C7AE        Zum nachsten Befehl
2025 JSR $CD8A        Numerischen Term auswerten
2028 JSR $D7F7        in Integerzahl umwandeln u. nach $14,$15
202B JMP $C8A3        BASIC-Zeilenadresse berechnen
```

Bild 4.9.3

PROGRAMM-EINZELHEITEN

1.) Name: JUMP/VARSUB

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Basic-Loader

4.) Anzahl der Bytes
des Basic-Loaders: 1167
des M-PGMs: 46

5.) Benutzte Variablen: s. Kap. 5.2

6.) Listings
des Basic-Loaders: Bild 4.9.1
des M-PGMs: Bild 4.9.3

7.) Erläuterungen zum Basic-Loader werden in Kap. 5.2 gegeben und zum M-PGM implizit im gut kommentierten Listing (Bild 4.9.3).

4.10 AUTONUMBER PER BASIC

AUTONUMBER sollte eigentlich Bestandteil des Betriebssystem sein, ist es aber beim VC-20 nicht. Halb so schlimm, darauf verzichten braucht man gar nicht, wenn man sich eines kleinen Hilfs-Programms bedient, was noch nicht einmal ein M-PGM sein muß, weil beim automatischen Numerieren von Zeilen keine Geschwindigkeitsanforderungen gestellt werden.

Das Programm AUTONUMBER (s. Bild 4.10.1) hat zwei Eigenschaften:

- Es können die Anfangszeilennummer und der Zeilennummernabstand eingegeben werden, auch zwischendurch geändert werden. Wenn keine Eingaben gemacht werden, werden die Nummern ab 10 im Abstand 10 hochgezählt.

- Wird die Programmeingabe irgendwann unterbrochen, so merkt sich das Programm die zuletzt eingegebene Zeilennummer und fährt mit der richtigen Nummer nach erneutem Aufruf fort.

In Verbindung mit dem Programm VC-KEEP (s. Kap. 4.2) können Sie AUTONUMBER ans RAM-Ende "wegpacken" und nach Belieben in den Arbeitsspeicher holen (FETCH-Funktion), wenn Sie es brauchen.

BEDIENUNGSHINWEISE

AUTONUMBER ist in Basic geschrieben und nimmt den Zeilennummernbereich von 62000-62011 ein. Das heißt, es ist darauf zu achten, daß man bei seiner Programmeingabe nicht aus Versehen auf eine dieser Nummern stößt. Falls Sie beabsichtigen die Zeilennummern von AUTONUMBER durch RENUMBER zu ändern, so gelingt dies nur teilweise: das PRINT"RUN62008" wird durch RENUMBER nicht verändert, muß man also "von Hand" umändern.

Bevor ein Programm eingetippt werden soll, bei dem man sich für den Einsatz von AUTONUMBER entschlossen hat, muß letzteres erst in den Arbeitsspeicher geladen werden. Aufgerufen wird es ganz normal per RUN.

Sie werden daraufhin nach der Anfangszeilennummer gefragt. Beim RETURN-Drucken ohne Zahleneingabe wird die Nummer 10 unterstellt. Falls Sie es gewohnt sind, mit einer anderen Zeilennummer zu beginnen und sich ebenfalls die anfängliche Eingabe sparen wollen, so wechseln Sie in Zeile 62000 einfach das ZN=10 entsprechend aus.

```
62000 ZN=10:INPUT"ZEILENNUMMER:",ZN
62001 DF=10:INPUT"ABSTAND:",DF:POKE253,DF
62002 HI=ZN/256:POKE251,ZN-HI*256:POKE252,HI:PRINTMID$(STR$(ZN),2)
62003 GETE$:POKE204,0:IFE$=""THEN62003
62004 IFASC(E$)=133THENPRINT:END
62005 PRINTE$;:IFE$<>CHR$(13)THEN62003
62006 PRINT"RUN62008":FORI=631TO634:POKEI,145:NEXT
62007 POKEI,13:POKEI+1,13:POKE198,6:END
62008 PRINTCHR$(145)CHR$(145):FORI=1TO3
62009 PRINT"          ":NEXT:PRINTCHR$(145)CHR$(145)CHR$(145);
62010 DF=PEEK(253)
62011 ZN=PEEK(251)+256*PEEK(252)+DF:GOTO62002
```

Bild 4.10.1

Danach ist die Eingabe des Zeilennummernabstands vorgesehen. Es werden hierbei nur Werte zwischen 1 bis 255 erwartet. Beim RETURN-Drücken ohne Zahleneingabe wird der Abstand 10 unterstellt. Falls Sie an einem anderen Abstand gewöhnt sind und sich ebenfalls eine Eingabe sparen wollen, so wechseln Sie in Zeile 62001 einfach das DF=10 entsprechend aus.

Sie sehen dann die gewünschte Zeilennummer am Anfang der Zeile und Sie können mit der Programmeingabe beginnen. Sobald Sie eine eingegebene Zeile per RETURN bestätigen, sehen Sie sofort die folgende Zeilennummer in der nächsten Zeile. Daß vielleicht ab und zu ein eingegebenes Zeichen in der Revers-Darstellung verbleibt, ist schlimmstenfalls rein optisch von Nachteil, im Programmspeicher selbst wird es ordnungsgemäß abgespeichert, wovon Sie sich später durch LIST überzeugen können.

Wenn Sie eine Unterbrechung bei der Eingabe des Programms wünschen, dann drücken Sie nicht ein weiteres RETURN, sondern die Funktionstaste F1. Wollen Sie irgenwann das Programm weitereintippen, so steigen Sie wieder ein mit:

RUN 62011 (Return)

Die in der Reihenfolge richtige nächste Zeilennummer wird dann am Anfang der Zeile zu sehen sein. Es könnte natürlich auch sein, daß Sie mit anderen Werten - Zeilennummer und Abstand - fortfahren möchten. In diesem Fall verwenden Sie:

RUN 62000 (Return)

Wenn Sie schließlich mit der Programmeingabe ganz fertig sind, so löschen Sie nach der Programmbeendigung über F1 die AUTONUMBER-Zeilen 62000-62011.

PROGRAMM-EINZELHEITEN

1.) Name: AUTONUMBER

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Betriebssystem-Zusatzroutine

4.) Anzahl der Bytes: 363

5.) Benutzte Variablen:

ZN	Zeilennummer
DF	Nummerndifferenz
H%	Hilfsvariable
E$	Eingegebenes Zeichen
I	Schleifenzähler

Benutzte Adressen:

#251,252	Zwischenspeicher für Zeilennummer (L,H)
#253	" " Nummerndifferenz

6.) Listing des PGMs: Bild 4.10.1

7.) Erläuterungen zum PGM:
Das Programm ist an sich leicht zu durchschauen. Hervorzuheben sind allerdings die Zeilen 62007 und 62008, worin über das HineinPOKEn in den Tastaturpuffer das automatische sich Aufbauen des Programms ermöglicht wird.

4.11 DUMP PER BASIC

DUMP sollte wie AUTONUMBER zu einem Basic-Betriebssystem dazugehören. Auch hier hat der Hersteller des VC-20 Kosten eingespart - andererseits kann man sich ja wirklich nicht über den VC-20-Preis beschweren. Aber auch in diesem Fall ist es relativ einfach, den VC-20-Funktionsvorrat um die Funktion DUMP zu erweitern.

Das Programm DUMP (s. Bild 4.11.1) ist ein kleines Basic-Hilfsprogramm, welches als Anhängsel zu jedem Basic-Programm hinzugefügt werden kann, vorausgesetzt, die Zeilennummern des Programms überschneiden sich nicht mit denen des Hilfsprogramms. Die Zeilennummern können jedoch ohne Bedenken per RENUMBER-Funktion (s. Kap. 4.8) umgeändert werden. Bevor Sie ein neues Programm eintippen, laden Sie zuerst DUMP (zusammen mit AUTONUMBER ist es auch verträglich). Mithilfe VC-KEEP (s. Kap. 4.2) allerdings können Sie jedes beliebige Basic-Programm Ihrer Programmbibliothek um DUMP erweitern.

Wenn Sie DUMP während eines Programmlaufs oder danach aufrufen, werden Ihnen sämtliche Variablen mit ihren augenblicklichen Werten und alle Funktionensnamen aufgezeigt, die währrend des Programmlaufs benutzt worden sind.

BEDIENUNGSHINWEISE

DUMP muß, wie schon oben beschrieben, als Anhängsel in demjenigen Programm vorhanden sein, dessen korrekte Abarbeitung Sie mithilfe der DUMP-Funktion überprüfen wollen. Ist einmal ein Programmlauf aufgrund eines Fehlerfalls, Abstoppen per Drucken der STOP-Taste oder per STOP bzw. END-Befehl beendet, dann rufen Sie die Anzeige sämtlicher Variablen einfach auf durch:

GOSUB 63000 (Return)

Falls eine Weiterbearbeitung des Programms von einer anderen Stelle (also nicht von Anfang an) beabsichtigt ist, dann darf dies nicht per GOTO-Befehl, sondern muß geschehen per:

RUN (gewünschte Zeilennummer) (Return)

```
63000 VV=PEEK(45)+256*PEEK(46)-7:FA=PEEK(47)+256*PEEK(48)+49
63001 H1=0:H1%=0:H1$=""
63002 PRINT:VV=VV+7:IFVV=FA-0*63THENRETURN
63003 N1=PEEK(VV):N2=PEEK(VV+1)
63004 ON(SGN(N1-127)*2+SGN(N2-127)+5)/2GOTO63005,63006,63008,63009
63005 GOSUB63010:PRINT"=";:ZZ=7:GOSUB63011:PRINTH1;:GOTO63002
63006 GOSUB63010:PRINT"$="CHR$(34);:ZZ=5:GOSUB63011:PRINTH1$CHR$(34);
63007 GOTO63002
63008 PRINT"FN ";:GOSUB63010:GOTO63002
63009 GOSUB63010:PRINT"%=";:ZZ=6:GOSUB63011:PRINTH1%;:GOTO63002
63010 PRINTCHR$(N1AND127);CHR$(N2AND127);:RETURN
63011 FORLV=2TO6:POKEFA-ZZ*7+LV,PEEK(VV+LV):NEXT:RETURN
```

Bild 4.11.1

Wie Sie dem Beispiel entnehmen konnen. ist der Aufruf GOSUB 63000 auch im Programm moglich.

Zur besseren Erläuterung gehen wir von einem Beispiel aus. Laden Sie das Programm DUMP und fügen Sie die Programmzeilen aus Bild 4.11.2 hinzu. Dann starten Sie es mit RUN und erhalten als Resultat die Variablenliste gemäß Bild 4.11.3. Falls Sie dies mit einem VC-20 in GV oder 3K-V ausprobieren, sind die Werte in den Variablen ab VV andere. Diese sind ohnehin nur Hilfsvariablen, die zum Unterprogramm DUMP gehören.

```
10 A=6
20 BB%=5
30 DEF FN CC(X)=X*A
40 DA$="TEST"
50 BB%=FNCC(BB%)
60 GOSUB 63000:END
```

Bild 4.11.2

```
A= 6
BB%= 30
FN CC
X= 0
DA$="TEST"
VV= 5138
FA= 5201
H1= 5201
H1%= 30
H1$="TEST"
N1= 78
N2= 50
ZZ= 7
LV= 3
```

Bild 4.11.3

Wenn Sie die Ausgabe der Variablen ab VV bis LV vermeiden wollen, dann müssen Sie nur im Unterprogramm DUMP in Zeile 63002 das "0*63" durch "1*63" ersetzen.

WIE WERDEN VARIABLEN GESPEICHERT ?

Diese Frage stellt sich zwangslaufig, will man die Funktionsweise des kleinen Programms DUMP verstehen. Das COMMODORE-Basic unterscheidet 4 verschiedene Variablentypen. Nach Abspeicherung erkennt das Betriebssytem den jeweiligen Typ an dem gesetzten bzw. nicht gesetzten Bits 7 der beiden Namenszeichen (NZ). Bei zwei Namenszeichen mit je zwei Moglichkeiten ergibt dies folglich insgesamt 4 unterscheidbare Falle. Variablennamen werden also grundsatzlich 2 Bytes Speicherplatz zugeorndet; wenn Sie einer Variablen nur einen Buchstaben spendieren (z.B.: A=5), so ergänzt das Betriebssystem von selbst diesen Namen zu zwei Zeichen, wobei letzteres Byte den Wert 0 erhalt, bzw. 128 bei gesetztem Bit 7.

Bit 7 von NZ1	Bit 7 von NZ2	Variablentyp
0	0	Gleitkomma
0	1	String ($)
1	0	Funktion (FN)
1	1	Integer (%)

Bild 4.11.4

Bei welchem Variablentyp welches Bit 7 gesetzt ist, veranschaulicht Bild 4.11.4 (NZ=Namenszeichen).

Neben den 2 Bytes fur den Variablennamen und -typ werden den Variablen vom Typ Gleitkomma, String und Integer noch weitere 5 Bytes zugewiesen, die deren Inhalt speichern sollen. Die Bilder 4.11.5a,5b,5c geben Auskunft daruber, wie das geschieht. Wahrenddessen die genannten Variablentypen insgesamt 7 Bytes an Speicherplatz verbrauchen, benötigt die Funktionsvariable 14 Bytes.

Adr. (L) und Adr. (H) bedeuten die LOW- und HIGH-Teile derjenigen Anfangsadresse, ab der der String mit der in Byte 3 angegebenen Länge abgespeichert ist. Die zwei letzten Bytes haben keine Funktion und sind generell auf 0 gesetzt. Wir konnen also hieraus er-

kennen, daß ein String (vom Namen abgesehen) 3 Bytes zu Verwaltungszwecken braucht. Bei der Abspeicherung von String-Feldern wird diese Tatsache vom Betriebssystem optimal berücksichtigt, indem jedes einzelne Feldelement mit einem Speicherbedarf von 3 Bytes auskommt, die String-Inhalte nicht einbezogen. Bei den Feldern steht am Anfang noch der Name und die Dimension.

Bild 4.11.5a: Gleitkomma-Variablentyp

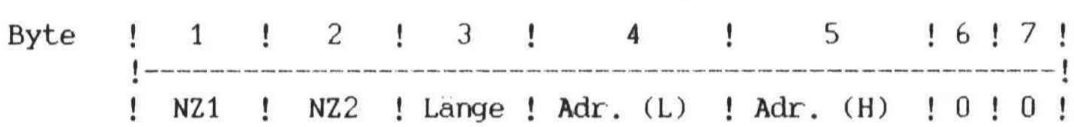


Bild 4.11.5b: String-Variablentyp

Bild 4.11.5c: Integer-Variablentyp

Die HIGH- und LOW-Teile beziehen sich auf den Wert der Integer-Variablen, wobei hierbei darauf zu achten ist, daß Integer-Variablen Werte von -32768 bis +32767 annehmen können. Das heißt, zwischen dem Variablenwert und den HIGH/LOW-Teilen läßt sich die folgende Beziehung herstellen:

```
Variablenwert = LOW + 256*(HIGH - 2*(HIGH AND 128))
```

Die drei letzten Bytes haben keine Funktion und sind generell auf 0 gesetzt. Ähnlich wie bei der String-Variablen können wir auch hier erkennen, daß eine Integer-Variable (vom Namen abgesehen)

nur 2 Bytes zur Speicherung ihres Wertes braucht. Bei der Abspeicherung von Integer-Feldern wird dieser Tatsache optimal Rechnung getragen, indem jedes einzelne Feldelement tatsachlich nur 2 Bytes einnimmt (auch hier stehen am Anfang der Name und die Dimension). Weil dagegen ein Gleitkomma-Feldelement 5 Bytes zur Wertspeicherung benotigt, wird dem Programmierer immer wieder geraten, nach Moglichkeit Integer-Felder anstatt Gleitkomma-Felder zu benutzen; das geht naturlich nicht immer.

Byte	1	2	3	4	5	6	7
	NZF1	NZF2	AD(L)	AD(H)	AA(L)	AA(H)	ASCD1
Byte	8	9	10	11	12	13	14
	NZA1	NZA2	Wert des Funktions-Arguments				

Bild 4.11.5d: Funktions-Variablentyp

Wie oben bereits erwähnt, nimmt eine Funktions-Variable 14 Bytes ein. Wofür die einzelnen Bytes dienen, zeigt Bild 4.11.5d. Die Abkürzungen haben die folgende Bedeutung:

NZF1,NZF2	Namenszeichen der Funktions-Variablen
AD(L),AD(H)	Anfangsadresse der Funktions-Definition
AA(L),AA(H)	Anfangsadresse des Argutmentwertes
ASCD1	ASCII-Code des 1. Zeichens der Funktions-Definition
NZA1,NZA2	Namenszeichen der Argument-Variablen

Folgendes Beispiel dient einer ergänzenden Erläuterung. Gehen wir von dem Einzeiler-Programm aus:

```
10 DEF FN AB(CD)=EF
```

welches in der Relation gem. Bild 4.11.6 zu den absoluten Adressen steht (AV angenommen, Adressen und Inhalte in HEX).

In diesem Beispiel ist:

NZF1,NZF2	Der Funktions-Name AB (ASCII-Code von "A" mit Bit 7 = 1 gem. Bild 4.11.4)
AD(L),AD(H)	Adresse $1210, weil ab dort die Funktion definiert wird
AA(L),AA(H)	Adresse $121E, weil ab dort der Wert der Variablen CD gespeichert ist
ASCD1	Der ASCII-Code von E = $45, weil "E" das 1. Zeichen der Definition "EF" ist
NZA1,NZA2	Der Name der Argument-Variablen ist CD

Adresse:	1201	1202	1203	1204	1205	1206	1207	1208	1209	120A	120B
Inhalt:	13	12	0A	00	96	20	A5	20	41	42	28
Bedeutung:	(Link)		10		DEF		FN		A	B	(

Adresse:	120C	120D	120E	120F	1210	1211	1212	1213	1214
Inhalt:	43	44	29	B2	45	46	0	0	0
Bedeutung:	C	D	)	=	E	F	(PGM-Ende)		

Bild 4.11.6

Folgerichtig müssen nach RUN dieses Einzeiler-Programms im Speicher die Werte gem. Bild 4.11.7 stehen.

Adresse:	1215	1216	1217	1218	1219	121A	121B
Inhalt:	C1	42	10	12	1E	12	45

Adresse:	121C	121D	121E	121F	1220	1221	1222
Inhalt:	43	44	0	0	0	0	0

Bild 4.11.7

PROGRAMM-EINZELHEITEN

1.) Name: DUMP

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Betriebssystem-Zusatzroutine

4.) Anzahl der Bytes: 414

5.) Benutzte Variablen:

VV	Anfangsadresse der Variablen
FA	" " Felder
H1	Hilfsvariable zwecks Zwischenspeicherung der Gleitkomma-Variablen-Werte
H1%	Hilfsvariable zwecks Zwischenspeicherung der Integer-Variablen-Werte
H1$	Hilfsvariable zwecks Zwischenspeicherung des String-Variablen-Inhalts
N1	ASCII-Code des 1. Namenszeichen
N2	" " " 2. Namenszeichen
ZZ	Zeigervariable zu H1, H1%, H1$
LV	Laufvariable

6.) Listing: Bild 4.11.1

7.) Erläuterungen zum Unterprogramm DUMP:

Zeile 63000:
Bestimmung von Variablen- (VV) und Feldanfang (FA)

Zeile 63001:
Initialisierung der Hilfsvariablen (Platzreservierung)

Zeilen 63002 - 63003:
Zur nächsten Variablen gehen (VV=VV+7) und deren Namenszeichen bestimmen (63003)

Zeile 63004:
Der Ausdruck zwischen ON und GOTO ist die Formel für die Zuordnung gemaß Bild 4.11.4 zu den Werten 1,2,3,4, welche über die ON-Anweisung zu den angegebenen Zeilen führen

Zeilen 63005 - 63007 und 63009:
Alle drei Zeilen arbeiten im Prinzip gleichermaßen. Sie geben unter Zuhilfenahme des UP-Einzeilers 63010 den Variablen-Namen aus, POKEn dann bei Hinzuziehung vom UP-Einzeiler 63011 die restlichen 5 Bytes der Variablen in die Hilfsvariable gleichen Variablentyps (FA-ZZ*7 ist die Anfangsadresse der entsprechenden Hilfsvariablen)

Zeile 63008:
Es wird der Funktions-Name ausgegeben

Zeilen 63010 - 63011:
s. Erläuterung für Zeilen 63005 ...

5

Maschinensprache-Programmierhilfen

5.1 DATA-ERZEUGER FÜR MASCHINENPROGRAMME

Des öfteren stellt sich dem Maschinenprogrammierer das Problem, für sein entwickeltes Maschinenprogramm einen Basic-Loader (s. Kap. 5.2) zu erstellen. Zur Abspeicherung eines M-PGMs über ein solches Ladeprogramm ist es notwendig, die einzelnen M-PGM-Codes in Form von Dezimalzahlen in DATA-Anweisungen präsent zu haben. Die Arbeit ein z.B. 500 Byte langes M-PGM in die Form von DATA-Zeilen zu bringen sollte man nicht unterschätzen.

Wir Autoren sind für die meisten in diesem Buch veröffentlichten M-PGM-Basic-Loader einen bequemeren Weg gegangen. Wir haben uns einfach ein Programm "gestrickt", welches uns diese mühsame Arbeit abgenommen hat.

BEDIENUNGSHINWEISE

Das Programm DATA-ERZEUGER (s. Bild 5.1.2) muß exakt eingetippt werden (falls Sie es nicht vom Verlag auf Kassette oder Diskette bezogen haben) und darf nicht verändert werden. Dann startet man es mit RUN.

Es werden hintereinander 3 Eingaben Ihrerseits verlangt:

- Zuerst werden Sie nach der Anfangs-Zeilennummer gefragt, bei der die DATA-Zeilen beginnen sollen. Dieser Wert muß >940 sein. Wir z.B.haben uns daran gewöhnt, die Zeilennummern mit 1000 beginnen zu lassen. Ab diesem Wert werden die Zeilennummern in 10er-Schritten hochgezählt.

- Dann muß man die dezimale Anfangsadresse des in DATA-Zeilen umzuwandelnden M-PGM eingeben. Liegt dieses z.B. im Speicher ab Adresse $2000, so ware die Eingabe 8192 erforderlich.

- Last not least muß noch die Frage nach der Länge des M-PGMs beantwortet werden. Nimmt z.B. das M-PGM den Speicherraum $2000-2012 (hexadezimal !!) ein, wobei das letzte M-PGM-Byte in Adresse $2012 angenommen wird, so würde man als Länge die Zahl (dezimal !!) 19 eingeben.

Falls übrigens irgendein Fehler bei den Eingaben passiert ist, ist es zwecklos, nach Betätigung der STOP-Taste das PGM erneut zu starten, weil dann schon einige PGM-Zeilen automatisch gelöscht sind. Es bleibt in diesem Fall nur das erneute Laden übrig.

Nach Drücken der RETURN-Taste im Anschluß an die letzte Eingabe findet ein emsiges Treiben auf dem Bildschirm statt. Man kann das Erzeugen der DATA-Zeilen richtiggehend verfolgen. Zum Abschluß löscht sich das PGM seine nicht mehr nötigen PGM-Zeilen von allein, so daß als Resultat nur noch die geforderten DATA-Zeilen übrigbleiben.

Zeigen wir doch einfach ein kleines Beispiel: Ab Adresse $EA8D = #60045 gibt es ein kleines VC-20-Betriebssystem-Unterprogramm (s. Bild 5.1.1), das für das Löschen einer Zeile sorgt (s. auch Anhang A.4).

```
EA8D A0 15     LDY #$15
EA8F 20 7E EA  JSR $EA7E
EA92 20 B2 EA  JSR $EAB2
EA95 A9 20     LDA #$20
EA97 91 D1     STA ($D1),Y
EA99 A9 01     LDA #$01
EA9B 91 F3     STA ($F3),Y
EA9D 88        DEY
EA9E 10 F5     BPL $EA95
EAA0 60        RTS
```

Bild 5.1.1

Mittels DATA-ERZEUGER stellen wir wie folgt DATA-Zeilen für dieses Unterprogramm her:

- Wir geben z.B. 1000 als Anfangszeilennummer ein.
- Die Frage nach der Anfangsadresse beantworten wir mit 60045.
- Als Länge geben wir 20 ein.

Als Resultat erhalten wir:

```
1000  DATA 160,21,32,126,234,32,178,234,169,32,145,209,169,1,145,
           243
1010 DATA  136,16,245,96
```

Um einen besseren Bezug zum ehemaligen M-PGM zu erhalten, werden immer 16 Bytes zu einer DATA-Zeile zusammengefaßt.

PROGRAMM-EINZELHEITEN

1.) Name: DATA-ERZEUGER

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Programm-Erzeuger

4.) Anzahl der Bytes: 912

5.) Benutzte Variablen:

BA	Nummer der DATA-Zeile
AA	Momentane M-PGM-Adresse
N	Anzahl der M-PGM-Bytes
EA	M-PGM-Endadresse
A	M-PGM-Code
A$	Zahl A als String
SZ,M,N	Hilfsvariablen
F1	Flagvariable
I	Schleifenzähler

6.) Listing: Bild 5.1.2

7.) Erläuterungen zum Programm:

Das Kernstück dieses Programm ist die Zeile 800. In dieser Zeile wird der Tastaturpuffer (ab Adresse #631) mit den Steuerzeichen CURSOR HOME (ASCII-Code #19) und 4 RETURN's (ASCII-Code #13) gefüllt. Außerdem muß dem Betriebssystem noch "gesagt" werden, daß insgesamt 5 Zeichen im Tastaturpuffer auf Ihre Abarbeitung warten (POKE 198,5).

```
10 REM  VC20-MASCHINENCODE-DATA-ERZEUGER
20 DATA0
30 DATA0
40 DATA1000
50 DATA0
60 DATA20,110,30,111,40,112,10,100,180,190,200,700,900,920,930,940,117,117,60,71
0
70 DATA70,50,116
80 DATA80,910,910
90 DATA90,730,800
100 PRINTCHR$(147)
110 READAA
111 READEA
112 READBA
116 READSZ
117 IFSZTHEN710
120 PRINT"ZEILENNUMMER DER"SPC(6)"1. DATA-ZEILE (>940):"
130 INPUTBA:PRINTCHR$(147);:GOSUB900:PRINT120:PRINT130:GOTO800
140 PRINT"MASCHINENCODE-ANFANGS-ADRESSE (DEZIMAL):"
150 INPUTAA:PRINTCHR$(147);:GOSUB920:PRINT140:PRINT150:GOTO800
160 PRINT"ANZAHL DER MASCHINEN- CODE-BYTES:"
170 INPUTN:EA=AA+N-1:PRINTCHR$(147);30;"DATA";EA:PRINT160:PRINT170:GOTO800
180 GOSUB930:IFF1THEN700
190 PRINTCHR$(147);BA;"DATA";A$;:FORI=1TO15:GOSUB930:IFF1THEN700
200 PRINT",";A$;:NEXT:BA=BA+10:PRINT:GOSUB900:GOSUB920:GOTO800
700 PRINT:SZ=1:GOSUB910:GOTO800
710 FORI=1TOSZ:READN:NEXT:READM:PRINTCHR$(147);N:PRINTM:SZ=SZ+2:GOSUB910:GOTO800
730 PRINTCHR$(147);:FORI=1TO3:READN:PRINTN:NEXT
800 POKE631,19:POKE632,13:POKE633,13:POKE634,13:POKE635,13:POKE198,5:PRINT"RUN":
END
900 PRINT40;"DATA",BA:RETURN
910 PRINT50;"DATA";SZ:RETURN
920 PRINT20;"DATA";AA:RETURN
930 IFAA>EATHENF1=1:RETURN
940 A=PEEK(AA):A$=STR$(A):A$=RIGHT$(A$,LEN(A$)-1):AA=AA+1:RETURN
```

Bild 5.1.2

Welches Prinzip liegt hier vor? Dadurch, daß der Tastaturpuffer per Programm mit Zeichen gefüllt wird, wird dem VC-20 eine Eingabe seitens des Benutzers "vorgegaukelt". Am letzten Befehl der Zeile 800 (PRINT"RUN") ist erkennbar, daß sogar ein erneuter Programmstart auf diese Weise bewerkstelligt wird. Die einfachere Benutzung des Befehles RUN reicht in diesem Programm nicht aus, weil RUN sämtliche Variablen löscht. Da zudem noch durch o.g. Prinzip Programmzeilen gelöscht werden, sind die Werte der gegenwärtigen Variablen ohnehin verloren. Also müssen hier DATA-Zeilen als Variablen-Übergabe-Hilfsmittel fungieren. Das sind die Zeilen 20-50.

In den DATA-Zeilen 60-90 stehen die Nummern der Programmzeilen, die mittels o.g. Verfahrens im Laufe der Abarbeitung gelöscht werden. Übrigens darf schon allein aus diesem Grund keine Veränderung an den Programmzeilennummern vorgenommen werden. Zeilen werden einfach dadurch gelöscht, daß die ent-

sprechende Zeilennummer auf den Bildschirm geschrieben wird (PRINT-Befehl), und bei Aussprung aus dem Programm (Zeile 800) die Zeile mit der Zeilennummer mit RETURN (denn dieses Steuerzeichen befindet sich ja im Tastaturpuffer) bestatigt wird, was der DELETE-Funktion fur Programmzeilen gleichkommt.

Auf diese Weise wird sukzessiv das Programm "abgebaut" gemaß dem Prinzip: was nicht mehr gebraucht wird, kann geloscht werden, so daß schließlich nur noch die bloßen DATA-Zeilen mit den gewunschten M-PGM-Dezimalcodes ubrigbleiben.

5.2 EIN SPEZIELLER BASIC-LOADER

Ein Basic-Loader hat bekanntlich die Aufgabe, per Basic-Programm, welches ja am einfachsten zu laden ist und bequem sonstige für den Benutzer wichtige Informationen in Form von REM- oder PRINT-Anweisungen beherbergen kann, ein Maschinen-Programm möglichst an beliebiger Stelle des freien RAM-Bereichs zu generieren. Es gibt zwar inzwischen schon die verschiedenartigsten Basic-Loader, jedoch selten findet man einen, der die wichtigsten Vorzuge zugleich aufweist.

Mit dem in diesem Kapitel vorgestellten Basic-Loader (s. Bild 5.2.1) wurde versucht, einen solchen als universell einsetzbar zu schaffen. Er weist die folgenden Merkmale auf:

- Die M-PGMe sind in jedem beliebigem Speicherbereich generierbar, auch wenn vom Speicherbereich abhängige Sprung- bzw. Unterprogrammaufrufe vorhanden sind.

- Der Benutzer hat für die Wahl der Anfangsadresse des M-PGM zwei Moglichkeiten, die für die meisten praktischen Fälle ausreichen:
  - Angabe einer absoluten Anfangsadresse
  - Eingabe von "E" als Entscheidung dafur, daß das M-PGM am Ende des verfügbaren Basic-RAMs abgespeichert werden soll. Die errechnete Anfangsadresse wird angezeigt.

- Es erfolgt eine Meldung, wenn folgende Fehler auftreten:
  - Die Checksumme stimmt nicht mit der im Basic-Loader eingetragenen uberein, was passieren kann, wenn die DATA-Zeilen-Werte nicht richtig eingegeben worden sind.
  - Die ausgesuchte Anfangsadresse liegt im ROM-Bereich oder adressiert eine Speicherzelle, in der ein Wert nicht gehalten werden kann (z.B. RAM nicht vorhanden).

- An das eigentliche M-PGM werden einige Bytes drangehängt, die dazu dienen, die Existenz und Lage von M-PGMen auch über ein RESET hinaus erkennbar zu machen.

```
10 REM (NAME DES PGM)
20 REM (ZWECK DES PGM)
30 L=(LAENGE DES M-PGM)
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>00000THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 END
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS";SDCHR$(146):PRINT
280 PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
```

Bild 5.2.1

BEDIENUNGSHINWEISE

Bevor das Programm gemäß Listing in Bild 5.2.1 seine Dienste aufnehmen soll, muß es für das jeweilig zu ladende M-PGM erst vervollständigt werden:

- Die Zeilen 10 und 20 sprechen fur sich.

- In Zeile 30 muß die tatsachliche Lange des M-PGMs angegeben werden.

- Falls die Einsprungadresse nicht mit der Anfangsadresse des M-PGMs ubereinstimmt, muß die letzte Anweisung der Zeile 150 abgeandert werden: zu AD wird die Adressendifferenz zwischen Einsprung- und Anfangsadresse addiert.

- In Zeile 230 mussen die 5 Nullen durch die korrekte Checksumme ersetzt werden. Wie man die am schnellsten herausbekommt, erfahren Sie in den "Programm-Einzelheiten" unter 7.)

- Ab Zeile 280 konnen per PRINT-Anweisungen dem Benutzer wesentliche Bedienungsinformationen gegeben werden.

- Ab Zeile 1000 (z.B.) werden die DATA-Zeilen, welche die Dezimal-Werte der M-PGM-Codes enthalten, nachgetragen. Diese konnen ubrigens bequem mit dem DATA-Erzeuger aus Kapitel 5.1 automatisch erzeugt werden.
  Nun zu einer Besonderheit, die diesen Basic-Loader als recht vorteilhaft auszeichnen: fur die Maschinenprogrammierer ist die absolute Adressierung der Maschinensprache ja oft ein Greuel, weil sich derartige Programme nicht verschieben las-sen. Nur die wenigsten sind "relocatable" (s. Anhang A.2). Uber diesen Basic-Loader jedoch werden sie quasi-relocatable, d.h.: die M-PGMe enthalten zwar absolute Adressen, sie gehen aber in relativer Form in dem Loader ein. Praktisch sieht das folgendermaßen aus. Wir gehen von folgendem kleinen Beispiel-Programm aus:

```
        Adresse ($)
2010            LDA $A3
2012            JSR $2018
2015            JMP $C474
2018            AND #$02
201A            RTS
```

Diese 11 Bytes Programmcode wurden per DATA-Erzeuger aus Kapitel 5.1 in die folgende DATA-Zeile verwandelt werden:

```
1000 DATA 165,163,32,24,32,76,116,196,41,2,96
```

Allerdings können momentan noch diese Befehlscodes nur ab Adresse $2010=#8208 abgespeichert werden. Quasi-relocatable wird der Basic-Loader nach dem folgendem Prinzip:
Dort, wo im Programm absolute Adressen wie in Zeile 2012 des Beispiels vorkommen, wird die dezimale Differenz zwischen der absoluten Adresse und der Anfangsadresse des M-PGMs berechnet. Im Beispiel würden wir herausbekommen:

$2018 (aus JSR $2018) - $2010 (Anfangsadresse) = #8

Diese Zahl mit einem vorangehenden "L" (für LOW) tritt nun an die Stelle des momentan in der DATA-Zeile vorliegenden LOW-Werts 24 (der 4. Wert). Dort, wo der HIGH-Wert steht - also der 5. Wert - steht in Zukunft nur noch ein "H". Die DATA-Zeile in neuem Gewand sieht hernach so aus:

1000 DATA 165,163,32,L8,H,76,116,196,41,2,96

Natürlich sind mittels diesem Prinzip nur solche absolute Adressen zu bearbeiten, die zum Adreßraum des PGMs selbst gehoren. Also JMP $C474 muß unverandert bleiben, es ist ja ein Sprung in eine unveranderbare VC-20-Betriebssystemroutine.

Eine Sache ist hier allerdings noch zu beachten: das M-PGM muß unbedingt so geartet sein, daß bei umzuandernden absoluten Adressen die Reihenfolge LOW, HIGH, LOW, HIGH, etc. eingehalten wird. Wieviele Bytes zwischen diesen Bytes liegen, spielt keine Rolle.

Es gibt noch eine weitere Besonderheit bei diesem Basic-Loader: beim Generieren des M-PGMs werden namlich 7 Bytes an das eigentliche M-PGM angehängt (s. Bild 5.2.2). Die letzten 3 Bytes dieses Anhängsel bestehen aus dem ASCII-Code des Wortes "END" (Bit 7 ist gesetzt). Die zwei darunterliegenden Bytes stellen den Vektor zum Anfang des M-PGMs dar; er wird automatisch berechnet. Zwei weitere Bytes dienen eventuellen Zusatzanwendungen seitens des Benutzers, erhalten also mittels dieses Basic-Loaders keinen definierten Wert.

Die hier definierten 7 "Verwaltungsbytes" haben den Vorteil, daß auch nach einem RESET mittels besonderem Programm (s. Kap. 4.2) alle am RAM-Ende existierenden M-PGMe identifizierbar bleiben und

der Basic-RAM-Ende-Vektor ($0055,0056) (zum Schutz der M-PGMe) korrekt wiederhergestellt werden kann.

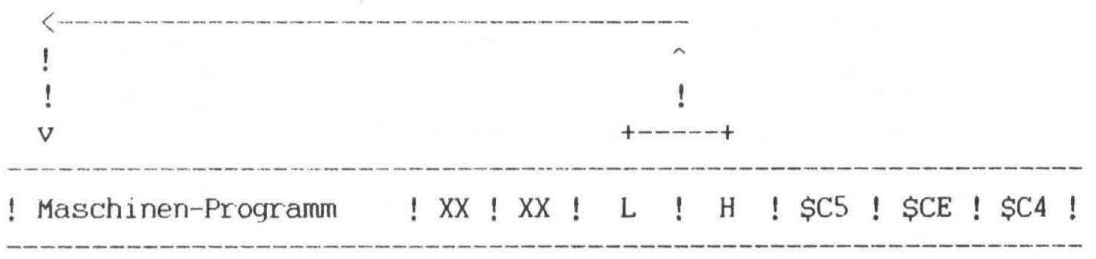


Bild 5.2.2

Die meisten der in diesem Buch vorgestellten M-PGMe bedienen sich dieses Basic-Loaders. Sie haben vielleicht schon die oben geschilderten Vorzuge kennengelernt oder werden sie noch erfahren.

PROGRAMM-EINZELHEITEN

1.) Name: BASIC-LOADER

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Ladehilfe für Maschinenprogramme

4.) Anzahl der Bytes: 941

5.) Benutzte Variablen:

L	Länge des M-PGMs
A$	Eingabe-Variable
SD	Vorgegebene Startadresse des M-PGMs; =0, wenn "E" eingegeben wird
EM	Adresse des Basic-RAM-Endes vor Abspeichern des M-PGMs
ER	Adresse des RAM-Endes
AD	Errechnete Startadresse des M-PGMs
AH%	HIGH-Teil von AD
AL	LOW-Teil von AD
I	Schleifenzähler
A	M-PGM-Code
CH	Checksumme

6.) Listing: Bild 5.2.1

7.) Erlauterungen zum Programm:

Zeile 30:
Hier wird die Länge des M-PGMs eingesetzt.

Zeilen 40 - 70:
Der Benutzer wird aufgefordert, entweder "E" oder eine Zahl einzugeben.

Zeile 80:
Die Eingabe (A$) wird per VAL-Funktion zur Zahl gemacht und SD zugewiesen. Falls dieser Wert verschieden von 0 ist, geht es weiter in Zeile 160.

Zeile 90:
Plausibilitatskontrolle, falls nicht "E" eingegeben wurde

Zeile 100:
Berechnung von: EM, dem augenblicklichen Basic-RAM-Ende
AD, dem zukunftigen Basic-RAM-Ende (nach Abspeichern des M-PGMs = M-PGM-Anfangsadresse
ER, dem RAM-Ende (Vektor #643,644)

Zeilen 110 - 140:
Falls noch kein M-PGM im RAM-Ende-Bereich abgespeichert worden ist (EM=ER), Fortsetzung in Zeile 130.
In Zeile 120 wird gepruft, ob am RAM-Ende die Kennzeichnung "END" (ASCII-Code mit Bit 7 = 1) vorliegt. Falls nein --> nach Zeile 150.
In die 3 letzten Speicherzellen des momentanen Basic-RAM-Ende wird die Kennzeichnung "END" hineingeschrieben. Die in Zeile 100 errechnete Anfangsadresse AD des zu ladenden M-PGMs wird um 7 Bytes heruntergesetzt und über das UP in Zeile 250 deren LOW- (AL) und HIGH-Teil (AH%) ermittelt. Dieser Vektor wird sodann in Zeile 140 direkt unterhalb der Kennzeichnung "END" gespeichert.

Zeile 150:
Das Basic-RAM-Ende ($0055,0056) wird auf den neuesten Stand gebracht, zeigt somit auf den Anfang des zu ladenden M-PGMs, damit es vor Überschreiben seitens Basic geschutzt ist. Die FRE-Funktion (AL ist hierbei nicht von Bedeutung) setzt entsprechend des aktuellen Vektors $55,56 auch die Vektoren $51,52 sowie $53,54 richtig. Die Startadresse SD erhält den Wert von AD, der Anfangsadresse des M-PGMs. Dies muß nicht immer so sein. Der Benutzer kann hier auch diese Einsprungadresse relativ zum M-PGM-Anfang angeben. Wenn die M-PGM-Einsprungadresse z.B. um 20 Bytes gegenuber der M-PGM-Anfangsadresse abweicht, mußte hier stehen: SD=AD+20

Zeile 170 - 220:
Schleife zur Generierung des M-PGMs. In Zeile 170 wird das nächste Datum (A$) der DATA-Anweisungen (z.B. ab Zeile 1000) gelesen. Wenn A$ eine Zahl ist, so wird in Zeile 180 der Variablen A diese Zahl zugewiesen, die Checksumme CH um sie erhöht und zur Zeile 210 gesprungen.
Ist das 1. Zeichen von A$ jedoch ein "L", so wird in Zeile 190 der Variablen AD diejenige Zahl zugewiesen, die auf das "L" folgt, die Checksumme CH um AD erhöht, zu AD der Wert von SD addiert und von AD der LOW- und HIGH-Teil über UP in Zeile 250 errechnet. A erhält den LOW-Wert und es folgt der Sprung zur Zeile 210.
Wenn obige beiden Möglichkeiten nicht galten, so muß A$ ein "H" sein, so daß A den vorher errechneten HIGH-Wert erhält.
In Zeile 210 schließlich wird das M-PGM-Byte in die geforderte Speicherzelle gePOKEd. Es wird hierbei geprüft, ob das POKEn erfolgreich war, indem der abgespeicherte Wert herausgePEEKed wird und mit dem abzuspeichernden verglichen wird. Wenn die Mühe vergebens war, erfolgt die Meldung in Zeile 220. Ansonsten wird der Schleifenlauf fortgesetzt.

Zeile 230:
Hier wird eine Checksummenprüfung vorgenommen. Der Programmierer erfährt den für sein jeweiliges abzuspeichernde M-PGM gültigen Checksummenwert einfach dadurch, daß er das M-PGM generiert (es folgt danach natürlich die Fehlermeldung aus Zeile 230) und per PRINT CH den korrekten Wert von CH abruft. Diesen trägt er dann einfach in Zeile 230 anstatt von 00000 ein.

Zeile 250:
LOW- und HIGH-Teil-Berechnung für den Wert in AD

Zeilen 260 - 280:
Dieses UP wird ausgehend von Zeile 160 angesprungen. In diesem Teil sollen alle nötigen Benutzerinformationen untergebracht werden, die für das jeweilige M-PGM gelten.

5.3 HEX-DEZ-KONVERTIERUNG

Es ist eine längst bekannte Tatsache: ohne Hexadezimalzahlen (s. Anhang A.2) geht's halt nicht bei der Maschinensprache-Programmierung. Natürlich kann man bei der Umrechnung von Dezimal-Zahlen in Hexadezimalzahlen (DEZ--->HEX) oder umgekehrt HEX--->DEZ den wahrscheinlich für den Einzelfall am muhelosesten Weg gehen, indem man die Berechnung einfach mit dem guten alten Taschenrechner vornimmt.

Wenn aber mehrere Berechnungen diesbezuglich notwendig werden, dann sollte das lieber ein Programm tun. Da sogar der COMMODORE-Monitor nicht über eine HEX-DEZ-Konvertierung verfügt und andere ähnliche Programme vielleicht diese ebenfalls nicht bieten, werden hier 2 Möglichkeiten gezeigt, eine solche Aufgabe zu erledigen:

- ein Basic-Unterprogramm, welches HEX- in DEZ-Zahlen umwandelt und für so manche Anwendungsfälle ausreichen dürfte

- ein M-PGM, welches unabhängig von jeglichem anderen Programm arbeitet und beide Konvertierungsrichtungen erlaubt. Es ist auch im Direkt-Modus benutzbar und gibt neben dem eigentlichen Resultat noch zusätzlich den dezimalen LOW- und HIGH-Anteil der jeweiligen Zahl an. Dies kann häufig von Nutzen sein, weil sämtliche Vektoren (z.B. Basic-Anfangs- oder Variablenanfangs-Vektor) im LOW/HIGH-Format eingegeben werden müssen.

1. KONVERTIERUNG PER BASIC

Das Unterprogramm in Bild 5.3.1 kann in jedes Basic-Programm eingebaut werden. Als HEX-Zahl in String E$ wird das UP aufgerufen und wird als DEZ-Zahl in E$ wieder verlassen. Falls E$ keinen Wert aufweist, die HEX-Zahl in E$ mehr als vierstellig ist oder eines der HEX-Ziffern nicht zum erlaubten Wertebereich gehört, wird der Variablen E$ der Wert "-1" zugewiesen, das als Fehlerkriterium im aufrufenden Programm vewertet werden kann.

```
1000 IFHX$=""THENHX$="0123456789ABCDEF"
1010 E=0:LE=LEN(E$):IFLE<1ORLE>4THEN1080
1020 FORI=1TOLE:HZ$=MID$(E$,I,1)
1030 FORJ=0TO15:IFMID$(HX$,J+1,1)=HZ$THEN1050
1040 NEXTJ
1050 IFJ>15THEN1080
1060 E=E*16+J
1070 NEXTI:E$=STR$(E):RETURN
1080 E$="-1":RETURN
```

Bild 5.3.1

```
0 REM UMSCHALTUNG IN GV
10 DATA 033C:REM ANFANGSADRESSE
20 DATA 78:REM SEI
30 DATA A2,00:REM LDX #$00
40 DATA 8E,81,02:REM STX $0281
50 DATA A9,10:REM LDA #$10
60 DATA 8D,82,02:REM STA $0282
70 DATA 20,CA,FD:REM JSR $FDCA
80 DATA 4C,32,FD:REM JMP $FD32
90 DATA@:REM ENDEZEICHEN
100 GOTO2000
500 IFD<0THENPRINT"FEHLER!":END
510 RETURN
1000 IFHX$=""THENHX$="0123456789ABCDEF"
1010 E=0:LE=LEN(E$):IFLE<1ORLE>4THEN1080
1020 FORI=1TOLE:HZ$=MID$(E$,I,1)
1030 FORJ=0TO15:IFMID$(HX$,J+1,1)=HZ$THEN1050
1040 NEXTJ
1050 IFJ>15THEN1080
1060 E=E*16+J
1070 NEXTI:E$=STR$(E):RETURN
1080 E$="-1":RETURN
2000 READE$:GOSUB1000:A=VAL(E$)
2010 READE$:IFE$="@"THENPRINT:PRINT"EINSPRUNG MIT:";PRINT"SYS";A:END
2020 GOSUB1000:D=VAL(E$):GOSUB500:POKEA+K,D:PRINTK+1;A+K;D:K=K+1:GOTO2010
```

Bild 5.3.2

Die folgenden Variablen werden benutzt:

E$	HEX-Zahl bzw. DEZ-Zahl nach Umwandlung
HX$	Liste der moglichen HEX-Ziffern
E	DEZ-Zahl, welche aus E$ gewonnen wird
LE	Anzahl der HEX-Ziffern in E$
I,J	Schleifenzähler

Bild 5.3.2 zeigt ein Anwendungsbeispiel für dieses Unterprogramm. Es stellt ein Basic-Loader dar, der im Gegensatz zu Dezimal-Zahlen (wie in allen anderen Basic-Loadern dieses Buches) den Objektcode in HEX in DATA-Zeilen beherbergt. Das M-PGM in diesem Beispiel (die Umschaltung in Grundversion, s. Kap. 6.3, Bild 6.3.1b) befindet sich in den DATA-Zeilen 10-90. Hierbei ist zu beachten, daß die Anfangsadresse ganz am Anfang (Zeile 10) und das Ende-Kennzeichen "Klammeraffe" am Ende steht (Zeile 90).

2. KONVERTIERUNG PER M-PGM

Das Maschinenprogramm VC-CONVERT (s. Bild 5.3.4 Teile 1-3) wird je nach Wunsch in einem beliebigen Speicherbereich generiert. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen überschreiben geschützt und sie können es auch dann aufrufen, wenn Sie beliebige andere Programme im Arbeitsspeicher haben. Weiterhin ist es koppelbar mit von Ihnen frei definierbaren Befehlen (s. Kap. 4.1).

BEDIENUNGSHINWEISE

Das M-PGM VC-CONVERT wurde in den im Kapitel 5.2 beschriebenen Basic-Loader (s. Bild 5.3.3) eingebunden. Detailinformation bezgl. diesem kann dort in Erfahrung gebracht werden. Der Basic-Loader wird normal mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das M-PGM generiert werden soll. Bei Eingabe von "E" wird es ans Basic-RAM-Ende gespeichert.

Nachdem RETURN gedrückt worden ist, werden auf dem Bildschirm diejenigen SYS-Befehle angezeigt, mittels denen die Konvertierungsroutine aufgerufen wird.

```
10 REM VC-CONVERT
20 REM DEZ-HEX- UND HEX-DEZ-ZAHLENKONVERTIERUNG
30 L=252
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>24736THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(157)",DEZ-TERM"CHR$(146):PRINT
275 PRINTCHR$(18)"SYS"SDCHR$(157)",$HEX-ZAHL"CHR$(146):PRINT
280 PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
1000 DATA32,115,0,201,36,240,44,201,35,208,3,32,115,0,32,L20
1010 DATAH,76,L206,H,32,138,205,32,247,215,162,0,32,L167,H,166
1020 DATA20,169,0,32,205,221,162,5,32,L167,H,166,21,169,0,32
1030 DATA205,221,96,169,255,72,160,4,32,115,0,240,20,136,48,248
1040 DATA162,15,221,L246,H,240,6,202,16,248,76,72,210,138,72,16
1050 DATA231,192,4,240,245,165,0,133,164,169,0,133,20,133,21,32
1060 DATAL134,H,133,20,144,8,32,L134,H,133,21,144,1,104,32,L26
1070 DATAH,162,10,32,L167,H,165,21,166,20,32,205,221,165,164,133
1080 DATA0,169,13,76,122,242,104,170,104,168,104,133,163,48,14,104
1090 DATA48,13,10,10,10,10,5,163,133,163,56,176,3,230,163,24
1100 DATA152,72,138,72,165,163,96,189,L226,H,240,6,32,122,242,232
1110 DATA208,245,96,189,L246,H,76,122,242,170,41,15,168,138,41,240
1120 DATA74,74,74,74,170,32,L179,H,152,170,32,L179,H,96,162,14
1130 DATA32,L167,H,165,21,32,L185,H,165,20,32,L185,H,162,18,76
1140 DATAL167,H,13,35,76,61,0,32,35,72,61,0,32,32,35,0
1150 DATA32,32,36,0,13,0,48,49,50,51,52,53,54,55,56,57
1160 DATA65,66,67,68,69,70
```

Bild 5.3.3

Beispielsweise haben Sie ein Gerät in Grundversion und entscheiden per Eingabe von "E" sich dafür, daß VC-CONVERT am Ende des Basic-RAM generiert wird. Falls nicht schon andere M-PGMe dort vorliegen, werden folgende SYS-Befehle angezeigt:

SYS 7418,(DEZ-Zahl) für die Umwandlung DEZ--->HEX
SYS 7418,$(HEX-Zahl) für die Umwandlung HEX->DEZ

5.3 HEX-DEZ-Konvertierung

Man kann vor die DEZ-Zahl auch das DEZ-Symbol "#" setzen oder sie kann durch eine Variable oder einen beliebigen Term ersetzt werden.

Z.B. wird nach: SYS 7418,12345 oder SYS 7418,#12345 (Return)
das Resultat: #L=57 #H=48 $3039 ausgegeben.

Nach: SYS 7418,$9600 (Return) erscheint: #L=0 #H=150 #38400

Vorausgehende Nullen bei HEX-Zahlen mit weniger als 4 Stellen konnen weggelassen werden. Die zusatzlichen Angaben vom LOW- (L) und HIGH-Anteil (H) sind in vielen Fällen recht hilfreich.

PROGRAMM-EINZELHEITEN

1.) Name: VC-CONVERT

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Basic-Loader

4.) Anzahl der Bytes
des Basic-Loaders: 1968
des M-PGMs: 262

5.) Benutzte Variablen: s. Kap. 5.2

6.) Listings
des Basic-Loaders: Bild 5.3.3
des M-PGMs: Bild 5.3.4 (Teile 1-3)

7.) Erlauterungen zum Basic-Loader werden in Kap. 5.2 gegeben und zum M-PGM implizit im Listing (Bild 5.3.4).

```
2000 JSR $0073       Hole naechstes Zeichen (nach Komma)
2003 CMP ##24
2005 BEQ $2033       Sprung, wenn "$"
2007 CMP ##23
2009 BNE $200E       Sprung, wenn nicht "#"
200B JSR $0073       Hole naechstes Zeichen
200E JSR $2014       Ausgabe "#L=...  #H=..."
2011 JMP $20CE
2014 JSR $C08A       DEZ-Term auswerten
2017 JSR $D7F7       Umwandl. in Integerzahl und nach $14,$15
201A LDX ##00
201C JSR $20A7       Ausgabe: "#L="
201F LDX $14
2021 LDA ##00        Ausgabe: LOW-Anteil von DEZ-Term
2023 JSR $DDCD
2026 LDX ##05
2028 JSR $20A7       Ausgabe: "#H="
202B LDX $15
202D LDA ##00        Ausgabe: HIGH-Anteil von DEZ-Term
202F JSR $DDCD
2032 RTS
2033 LDA ##FF        ##FF=Endezeichen
2035 PHA                  in Stapel legen
2036 LDY ##04        Y=Schleifenzaehler=4
2038 JSR $0073       Hole naechstes Zeichen
203B BEQ $2051       Sprung, wenn ":" oder Zeilenende
203D DEY             Y=Y-1
203E BMI $2038       Sprung, wenn Y<0, (Y-Schleifenende)
2040 LDX ##0F        X=Schleifenzaehler=15
2042 CMP $20F6,X     Vergleich mit HEX-Ziffern "0...F"
2045 BEQ $204D       Sprung, wenn gefunden
2047 DEX
2048 BPL $2042       X-Schleifenende
204A JMP $D248       Ausgabe: "ILLEGAL QUANTITY"
204D TXA             Ziffer in den Stapel
204E PHA
204F BPL $2038       Sprung: immer
2051 CPY ##04
2053 BEQ $204A       Sprung, wenn Y=4
2055 LDA $00         Zelle $00 retten
2057 STA $A4
2059 LDA ##00        Zellen $14,$15 mit $00 initialisieren
205B STA $14
205D STA $15
205F JSR $20d6       Berechne L-Byte der HEX-Zahl
2062 STA $14         nach $14
2064 BCC $206E       Sprung, wenn keine HEX-Ziffer mehr
2066 JSR $2066       Berechne H-Byte der HEX-Zahl
2069 STA $15         nach $15
206B BCC $206E       Sprung, wenn keine HEX-Ziffer mehr
206D PLA             Ende-Zeichen aus Stapel holen
206E JSR $201A       Ausgabe: "#L=... #H=..."
2071 LDX ##0A
2073 JSR $20A7
2076 LDA $15
2078 LDX $14
207A JSR $DWCD       Ausgabe DEZ-Zahl
```

Bild 5.3.4 (Teil 1)

```
2070 LDA $A4          Zeile $00 in Ausgangszustand versetzen
2072 STA $00
2074 LDA #$0D         CR ausgeben
2076 JMP $F27A
-------------------------------------------------------------
2079 PLA              Ruecksprungadresse aus Stack
207A TAX
207B PLA
207C TAY
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
207D PLA              HEX-Ziffer aus Stack und in $A3 retten
207E STA $A3
2080 BMI $2092        Sprung, falls Ende-Zeichen
2082 PLA              HEX-Ziffer aus Stack
2083 BMI $2092        Sprung, falls Ende-Zeichen
2085 ASL
2086 ASL              HEX-Ziffer = 1/2 Byte linksbuendig
2087 ASL                  setzen
2088 ASL
2089 ORA $A3          erstere HEX-Ziffer hinzunehmen
208B STA $A3          retten in $A3
208D SEC              CARRY gesetzt: evtl. noch mehr HEX-Ziff.
208E BCS $2080        Sprung: immer
2090 INC $A3          $A3 entnaelt #$00
2092 CLC              CARRY geloescht: keine HEX-Ziffern mehr
2093 TYA
2094 PHA              Ruecksprungadresse zurueck in stack
2095 TXA                  und Ergebnis in A laden
2096 PHA
2097 LDA $A3
2099 RTS
-------------------------------------------------------------
209A LDA $20E2,X      Ausgabe von durch X indizierte
209D BEQ $20B2            Teile der Tabelle 1
209F JSR $F27A
20A2 INX
20A3 BNE $209A
20A5 RTS
-------------------------------------------------------------
20A6 LDA $20F6,X      Ausgabe von durch X indizierte
20A9 JMP $F27A            Teile der Tabelle 2
20AC TAX
20AD AND #$0F         Ermittlung und Ausgabe der 2 HEX-Ziffern
20AF TAY                  des in A uebergebenen Bytes
20B0 TXA
20B1 AND #$F0
20B3 LSR
20B4 LSR
20B5 LSR
20B6 LSR
20B7 TAX
20B8 JSR $209E
20BB TYA
20BC TAX
20BD JSR $20B6
20C0 RTS
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
20C1 LDX #$0E         Ausgabe: "$"(HIGH-Byte)
20C3 JSR $20A7
20C6 LDA $13
20C8 JSR $20B9
20CB LDA $14          Ausgabe: "$"(LOW-Byte)
20CD JSR $20B9
20D0 LDX #$12
20D2 JMP $20A7
```

Bild 5.3.4 (Teil 2)

```
20E2 0D 23 4C 3D 00
20E7 20 23 48 3D 00          Tabelle 1
20EC 20 20 23 00 20
20F1 20 24 00 0D 00


20F6 30 31 32 33 34
20FB 35 36 37 38 39          Tabelle 2
2100 41 42 43 44 45
2105 46 00 FF 00 FF
```

Bild 5.3.4 (Teil 3)

5.4 SPEICHERBEREICHE VERSCHIEBEN, VERGLEICHEN UND FÜLLEN

Manchmal braucht der Programmierer die Moglichkeit, die Inhalte eines Speicherbereiches in einen anderen Bereich zu kopieren bzw. zu verschieben. Wenn es sich dabei um eine kleinere Anzahl von Bytes handelt, laßt sich dies per Basic-Schleife bei Einsatz von PEEK und POKE einfach und schnell bewerkstelligen. Diese Methode kostet jedoch recht viel Zeit, wenn es sich um z.B. ein 8 KByte langen Bereich handelt, und das kommt nicht so selten vor, wenn wir an die VC-20-Modulspiele denken.

Das hier beschriebene Programm (s. Bild 5.4.1) sorgt nicht nur furs Übertragen von Bytes (TRANSFER), sondern vermag auch die Inhalte von zwei Speicherbereichen miteinander zu vergleichen (COMPARE) oder auch einen Bereich mit einem vorgegebenem Wert zu fullen (FILL), dies alles in Sekundenbruchteilen.

Das Maschinenprogramm wird je nach Ihrem Wunsch in einem beliebigen Speicherbereich generiert. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen Überschreiben geschützt (es sei denn, Sie zerstoren es aus Versehen bei Verwendung der FILL-Funktion) und sie konnen es auch dann aufrufen, wenn Sie beliebige andere Programme im Arbeitsspeicher haben. Weiterhin ist es koppelbar mit von Ihnen frei definierbaren Befehlen (s. Kap. 4.1).

BEDIENUNGSHINWEISE

Das Programm wird normal mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das M-PGM generiert werden soll. Bei Eingabe von E wird es ans Basic-RAM-Ende gespeichert. Andere dort eventuell schon vorhandene M-PGMe werden nicht angetastet.

```
10 REM VC-TRANSFER/COMPARE/FILL
20 REM SCHNELLES UEBERTRAGEN/VERGLEICHEN/FUELLEN VON SPEICHERBEREICHEN
30 L=235
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>28543THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"UEBERTRAGEN MIT:"
270 PRINTCHR$(18)"SYS"SDCHR$(157)",QA,ZA,N"CHR$(146):PRINT
272 PRINT"VERGLEICHEN MIT:"
274 PRINTCHR$(18)"SYS"SD+6CHR$(157)",QA,ZA,N"CHR$(146):PRINT
277 PRINT"FUELLEN MIT:"
279 PRINTCHR$(18)"SYS"SD+12CHR$(157)",AA,EA,I"CHR$(146):PRINT
280 PRINT"QA=QUELLBEREICHANFANG ZA=ZIELBEREICHANFANG"
281 PRINT"N=ANZAHL DER BYTES    AA=ANFANGSADRESSE
283 PRINT"EA=ENDADR. (INCL.)    I=INHALT (0...255)":PRINT:PRINT
285 PRINT:PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
1000 DATA32,L18,H,76,L54,H,32,L18,H,76,L171,H,32,L18,H,76
1010 DATAL199,H,120,32,121,0,208,3,76,8,207,201,44,208,3,32
1020 DATA115,0,32,L95,H,132,163,133,164,32,L92,H,132,166,133,167
1030 DATA32,L92,H,160,0,96,56,165,163,229,166,165,164,229,167,144
1040 DATA52,32,L106,H,48,46,177,163,145,166,32,L79,H,208,242,230
1050 DATA163,208,2,230,164,230,166,208,2,230,167,96,32,253,206,32
1060 DATA138,205,32,247,215,132,168,133,169,96,165,168,208,2,198,169
1070 DATA198,168,36,169,96,24,165,168,72,101,166,133,166,165,169,170
1080 DATA101,167,133,167,24,104,101,163,133,163,138,101,164,133,164,165
1090 DATA163,208,2,198,164,198,163,165,166,208,2,198,167,198,166,32
1100 DATAL106,H,48,208,177,163,145,166,76,L143,H,32,L106,H,48,196
1110 DATA177,163,209,166,240,12,166,163,165,164,32,205,221,169,32,32
1120 DATA122,242,32,L79,H,208,228,165,169,240,3,76,72,210,166,168
1130 DATA56,165,166,229,163,133,168,165,167,229,164,133,169,144,236,138
1140 DATA145,163,32,L79,H,32,L106,H,16,245,96
```

Bild 5.4.1

Nach Drucken von RETURN erhalten Sie auf dem Bildschirm eine ausführliche Beschreibung über die Bedienungsweise des M-PGMs.

Falls Sie ein Rechner mit GV (Grundversion) besitzen und sich über die Eingabe "E" für das Generieren des M-PGMs am RAM-Ende entschieden haben, konnen Sie die 3 Funktionen TRANSFER, COMPARE und FILL in der folgenden Weise aufrufen:

TRANSFER durch Eingabe von: SYS 15743,QA,ZA,N
wobei QA die Anfangsadresse des zu verschiebenden Bereiches, ZA die Anfangsadresse des Zielbereiches und N die Anzahl der zu verschiebenden Bytes ist.

COMPARE durch Eingabe von: SYS 15749,QA,ZA,N
QA, ZA, N siehe bei TRANSFER

FILL durch Eingabe von: SYS 15755,AA,EA,I
wobei AA die Anfangs- und EA die Endadresse (einschl.) desjenigen Bereichs ist, der sämtlich mit dem Wert I gefüllt wird.

Die SYS-Adressen sind von Ihrer Wahl der M-PGM-Adresse oder bei "E" von der bereits am RAM-Ende sich befindenden anderen M-PGMe bzw. von der Speicherausbaustufe abhängig, wird aber automatisch ausgerechnet und Ihnen sofort bei Generierung angezeigt. Es ist ratsam, sich diese Adressen irgendwohin aufzuschreiben. Dies ist allerdings nicht notwendig, wenn Sie die 3 SYS-Sprünge indirekt über selbst definierte Befehle realisieren (s. Kap. 4.1).

PROGRAMM-EINZELHEITEN

1.) Name: VC-TRANSFER/COMPARE/FILL

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Basic-Loader

4.) Anzahl der Bytes
des Basic-Loaders: 2133
des M-PGMs: 235

5.) Benutzte Variablen: s. Kap. 5.2

```
2000 JSR $2012          Quelle, Ziel, Anzahl einlesen
2003 JMP $2036          SPrung, zur TRANSFER-Routine
2006 JSR $2012          Quelle, Ziel, Anzahl einlesen
2009 JMP $20AB          SPrung zur COMPARE-Routine
200C JSR $2012          Anfang, Code, Inhalt einlesen
200F JMP $20C7          SPrung zur FILL-Routine
2012 SEI
2013 JSR $0079          Momentanes Zeichen nach A
2016 BNE $201B          SPrung, wenn noch nicht Zeile zuende
2018 JMP $CF08          Ausgabe: "SYNTAX ERROR"
201B CMP #$2C
201D BNE $2022          SPrung, wenn momentane Zeile <> Komma
201F JSR $0073          Hole naechstes Zeichen
2022 JSR $205F          1. Zahl einlesen
2025 STY $A3            nach $A3,$A4
2027 STA $A4
2029 JSR $205C          2. Zahl einlesen (mit Komma-Check)
202C STY $A6            nach $A6,$A7
202E STA $A7
2030 JSR $205C          3. Zahl einlesen (mit Komma-Check)
2033 LDY #$00           Y = 0
2035 RTS
2036 SEC
2037 LDA $A3            ($A3,$A4) - ($A6,$A7)
2039 SBC $A6
203B LDA $A4
203D SBC $A7
203F BCC $2075          SPrung, wenn Quelladresse < Zieladresse
2041 JSR $206A          Zaehlerstand abfragen
2044 BMI $2074          fertig, wenn Zaehler unterschritten
2046 LDA ($A3),Y        Inhalt von ($A3),0
2048 STA ($A6),Y                        auf ($A6),0 uebertragen
204A JSR $204F          Vektoren $A3,$A4 und $A6,$A7 erhoehen
204D BNE $2041
204F INC $A3            Vektoren $A3,$A4
2051 BNE $2055
2053 INC $A4                    und $A6,$A7 erhoehen
2055 INC $A6
2057 BNE $205B
2059 INC $A7
205B RTS
205C JSR $CEFD          Komma-Check
205F JSR $CD8A          numerischen Term einlesen und
2062 JSR $D7F7          Integer-Umwandlung nach Y,A
2065 STY $A8            Zahl nach $A8,$A9 (Zaehler)
2067 STA $A9
2069 RTS
206A LDA $A8            Zaehler $A8,$A9 vermindern
206C BNE $2070
206E DEC $A9
2070 DEC $A8
2072 BIT $A9            Bit 7 - Check vom HIGH-Zaehlerbyte
2074 RTS
```

Bild 5.4.2 (Teil 1)

```
2075 CLC                    Addition: $A6,$A7 + $A8,$A9
2076 LDA $A8
2078 PHA                    ($A8) retten
2079 ADC $A6
207B STA $A6
207D LDA $A9
207F TAX                    ($A9) retten
2080 ADC $A7
2082 STA $A7
2084 CLC                    Addition: $A3,$A4 + $A8,$A9
2085 PLA                    ($A8)
2086 ADC $A3
2088 STA $A3
208A TXA                    ($A9)
208B ADC $A4
208D STA $A4
208F LDA $A3                Vektoren $A3,$A4
2091 BNE $2095
2093 DEC $A4                          und $A6,$A7 vermindern
2095 DEC $A3
2097 LDA $A6
2099 BNE $209D
209B DEC $A7
209D DEC $A6
209F JSR $206A              Zaehlerstand abfragen
20A2 BMI $2074              Sprung, wenn Zaehler unterschritten
20A4 LDA ($A3),Y            Inhalt ($A3),0 auf
20A6 STA ($A6),Y                              ($A6),0 uebertragen
20A8 JMP $208F              Sprung: immer
20AB JSR $206A              Zaehlerstand abfragen
20AE BMI $2074              Sprung, wenn Zaehler unterschritten
20B0 LDA ($A3),Y            Inhalt ($A3),0 mit
20B2 CMP ($A6),Y                              ($A6),0 vergleichen
20B4 BEQ $20C2              Sprung, wenn identisch
20B6 LDX $A3
20B8 LDA $A4
20BA JSR $DDCD              Vektor $A3,$A4
20BD LDA #$20                         und
20BF JSR $F27A                        SPACE ausgeben
20C2 JSR $204F              Vektor $A3,$A4 erhoehen
20C5 BNE $20AB              Sprung: immer
20C7 LDA $A9
20C9 BEQ $20CE              Sprung, wenn Bytewert < 256
20CB JMP $D248              Ausgabe: "ILLEGAL QUANTITY"
20CE LDX $A8                Bytewert nach X
20D0 SEC
20D1 LDA $A6                Differenz ($A6,$A7) - ($A3,$A4)
20D3 SBC $A3
20D5 STA $A8                                  nach $A8,$A9 (Zaehler)
20D7 LDA $A7
20D9 SBC $A4
20DB STA $A9
20DD BCC $20CB              Sprung, wenn Ende < Anfang
20DF TXA
20E0 STA ($A3),Y            Bytewert nach ($A3),0
20E2 JSR $204F              Vektor $A3,$A4 erhoehen
20E5 JSR $206A              Zaehler $A8,$A9 vermindern
20E8 BPL $20DF              Sprung, wenn noch nicht < 0
20EA RTS                    fertig
```

Bild 5.4.2 (Teil 2)

6.) Listings
des Basic-Loaders: Bild 5.4.1
des M-PGMs: Bild 5.4.2 (Teil 1 u. Teil 2)

7.) Erläuterungen zum Basic-Loader werden in Kap. 5.2 gegeben und zum M-PGM implizit im Listing (Bild 5.4.2).

6

Auswahl einiger Nutzprogramme

6.1 DATA SUCHEN

Das Programm "DATA SUCHEN" stellt Ihnen eine Maschinensprache-Routine zur Verfugung, mit deren Hilfe DATA-Zeilen nach vorgegebenen Begriffen durchsucht werden können. Alle Daten, die den vorgegebenen Begriff enthalten, werden nach Rückkehr aus dem M-PGM mit READ gelesen und einer Variablen zur Weiterverarbeitung übergeben. In unserem kleinen Basic-PGM, das übrigens nur als Beispiel dienen soll, wird das Datum, das den gesuchten Begriff enthalt, der Variablen SS$ übergeben und anschließend auf den Bildschirm dargestellt. Das PGM sucht solange nach dem vorgegebenen Begriff, bis der Klammeraffe (@) als DATA-Ende-Zeichen erkannt wird (Zeile 60000).

Der große Nutzen dieser Maschinensprache-Routine liegt in der Geschwindigkeit, mit der die DATA-Zeilen nach den vorgegebenen Begriffen durchsucht werden. Eine Suchroutine mit den gleichen Leistungsmerkmalen in Basic programmiert benötigt, abhängig von der Anzahl der Daten, etwa das 50- bis 100fache der Zeit. Testen Sie es selbst!

BEDIENUNGSHINWEISE

Bevor Sie das PGM starten, müssen die Daten eingegeben werden. In unserem Beispiel können die Zeilen 321 bis 59999 mit Daten beschrieben werden.

Wenn Sie dann das PGM starten, wird das M-PGM generiert und der Basic-Loader anschließend gelöscht. Danach verlangt der Computer die Eingabe des gesuchten Begriffs (max. 16 Zeichen), z.B. "ABC". Nach Beenden der Eingabe (Return) werden alle Daten, in denen die Buchstabenkombination "ABC" enthalten ist, auf dem Bildschirm dargestellt. Von einem gefundenen Datensatz gelangt man zum nachsten einfach durch Drücken der Leerzeichentaste. Wenn keine Daten (mehr) gefunden werden, erscheint der Hinweis "Nichts (mehr) gefunden". Man drückt wieder auf die Leerzeichentaste und kann mit erneuter Suchbegriffeingabe fortfahren.

Das PGM kann in zwei Arten verlassen werden:

- Wird man aufgefordert, ein Suchbegriff einzugeben, und druckt "Pfeil nach oben", so ist das Programm beendet.

- Wird ein gefundener Datensatz angezeigt und die "Pfeil nach oben"-Taste gedruckt, so wird das Programm zwar auch verlassen, es wird jedoch zusätzlich diejenige Zeile ausgeLISTet, worin der gefundene Datensatz vorkommt. Dies kann für Editierzwecke recht nützlich eingesetzt werden.

Das beendete Programm ist nun scheinbar unvollständig, weil - wie anfangs erwähnt - derjenige Teil des Programms gelöscht wird, der die schnelle Maschinensprache-Suchroutine generiert. Die Generierung ist allerdings so vorgenommen worden, daß sich dieses M-PGM zwischen dem Basic-PGM-Ende und der Adresse liegt, wohin der Variablenanfangs-Vektor (Adressen $45,46) zeigt. Somit kann das eben verlassene, evtl. auch editierte, PGM ohne Bedenken abgeSAVEd werden, denn das Such-M-PGM ist sicher im Programm verpackt.

PROGRAMM-EINZELHEITEN

1.) Name: DATA SUCHEN

2.) Ausbaustufe: beliebig

3.) Art des PGMs: "Elektronischer Karteikasten" mit schneller Suchroutine

4.) Anzahl der Bytes

des PGMs mit Basic-Loader-Teil:	1605
" " ohne " :	696
des M-PGMs:	256

5.) Benutzte Variablen:

A	Startadresse für SUCHEN
B	" " WEITERSUCHEN
I	Schleifenzähler
M	M-PGM-Code
S	Länge des gesuchten Begriffs
A$	Tastatur abfragen

S$	Gesuchter Begriff
SS$	Gefundender Datensatz
E$	Eingebenes Zeichen

Benutzte Zero-Page-Adressen:

$00	Lange des gesuchten Begriffs
$01,02	Adresse der vorigen Basic-Zeile
$03,04	" " aktuellen " "
$FB,FC	" " nächsten " "
$05	Zwischenspeicher furs Y-Register

```
1 DATA110,130,140,11,12,13,14,1
2 DATA15,16,17,18,19,20,21,2
3 DATA22,23,24,25,150,10,3,160
10 DATA165,43,133,3,165,44,133,4,160,0,177,43,133,251,200,177
11 DATA43,133,252,234,165,3,133,1,165,4,133,2,165,251,133,3
12 DATA165,252,133,4,160,0,177,3,133,251,200,177,3,133,252,200
13 DATA177,3,133,63,200,177,3,133,64,234,200,177,3,201,131,208
14 DATA211,136,132,5,234,230,5,164,5,162,0,234,200,177,3,240
15 DATA195,201,64,240,11,221,6,18,208,235,232,228,0,208,237,234
16 DATA136,177,3,201,44,240,31,201,131,208,245,234,160,2,177,1
17 DATA133,63,200,177,1,133,64,234,200,177,1,208,251,165,1,166
18 DATA2,133,3,134,4,234,166,4,165,3,132,5,24,101,5,144
19 DATA1,232,134,66,133,65,96,208,172,165,43,133,3,165,44,133
20 DATA4,234,160,0,177,3,133,251,200,177,3,133,252,165,3,133
21 DATA1,165,4,133,2,165,251,133,3,165;252,133,4,160,0,177
22 DATA3,133,251,200,177,3,133,252,200,177,3,197,63,208,222,200
23 DATA177,3,197,64,208,215,165,4,197,66,240,14,24,169,0,229
24 DATA3,24,101,65,133,5,198,5,208,173,24,165,65,229,3,133
25 DATA5,198,5,208,162,240,160,0,0,0,0,0,0,0,0,0
100 REM...16 STELLEN...*
110 POKE46,PEEK(46)+1:CLR
120 A=(PEEK(46)-1)*256+PEEK(45):B=A+153
130 FORI=1TO24:READM:NEXT
140 FORI=0TO255:READM:POKEA+I,M:NEXT:RESTORE
150 PRINTCHR$(147);:FORI=1TO8:READM:PRINTM:NEXT
160 POKE631,19:FORI=1TO9:POKE631+I,13:POKE198,10:NEXT:PRINT"RUN":END
170 S$="":PRINTCHR$(147):PRINT:PRINT:PRINT:INPUT"SUCHBEGRIFF";S$
180 S=LEN(S$):IFS<1ORS>16THEN120
190 IFS$="↑"THENEND
200 POKE0,S
220 FORI=1TOS:POKEPEEK(44)*256+PEEK(43)+4+I,ASC(MID$(S$,I,1)):NEXT:SYSA
230 READSS$:IFSS$="@"THEN250
240 PRINT:PRINT:PRINTSS$:GOSUB260:SYSB:GOTO230
250 PRINT:PRINT:PRINT"NICHT (MEHR) GEFUNDEN":GOSUB260:GOTO170
260 PRINT:PRINTCHR$(29)CHR$(18)"WEITER MIT TASTE"CHR$(146)
270 GETE$:IFE$=""THEN270
280 IFE$<>"↑"THENRETURN
290 PRINTCHR$(147)CHR$(145)"L↖"PEEK(63)+256*PEEK(64)
300 POKE631,19:POKE632,13:POKE198,2:END
60000 DATA@
```

Bild 6.1.1

6.) Listings
des PGMs mit Basic-Loader-Teil: s. Bild 6.1.1
" " ohne " : s. Bild 6.1.2
des M-PGMs: s. Bild 6.1.3

7a) Erlauterungen zum Basic-Programm (s. Bild 6.1.1):

Zeilen 1 - 3:
Zeilennummern, die nach Generierung des M-PGMs geloscht werden

Zeilen 10 - 25:
DEZ-Dump der M-PGMs

Zeile 100:
Die 16 Stellen hinter REM dienen als Ubergaberegister fur das M-PGM. Die Große des Ubergaberegisters ist willkurlich gewahlt worden. Sie kann durch Andern der Zeilen 100 und 180 dem jeweiligen Bedarf angepaßt werden.

Zeilen 120 - 160:
Basic-Loader fur das M-PGM; A = Startadresse SUCHEN, B = Startadresse WEITERSUCHEN; der Basic-Loader wird mittels der Zeilen 150 und 160 nach Generierung des M-PGMs geloscht. Dies funktioniert folgendermaßen:
Erst wird der Bildschirm geleert (CHR$(147)), dann werden die 8 Zahlen der DATA-Zeile 1 gelesen und in die ersten 8 Zeilen des Bildschirms geschrieben, in die 9. Zeile wird der Befehl RUN ausgegeben (am Ende der Zeile 160). In den Tastaturpuffer - die Speicherzellen #631-640 - werden 10 Zeichen abgelegt (mehr ist nicht moglich!): zu Anfang der ASCII-Code fur CURSOR HOME (#19), dann per Schleife 9mal CR (#13). Zum Schluß muß per POKEn der Zahl 10 in die Speicherzelle #198 dem VC-20 noch mitgeteilt werden, daß 10 Zeichen im Tastaturpuffer auf ihre Abarbeitung warten.
Was geschieht hierbei nun? Alle 8 Zeilen, die in DATA-Zeile 1 stehen werden geloscht, weil der VC-20 den vorgegebenen Zeichen des Tastaturpuffer gemaß mit dem Cursor zur 1. Zeile springt und dann 9mal die Taste "Return" von allein bestatigt. Da jedoch die auf dem Bildschirm stehenden Zeilennummern ohne Inhalt mit "Return" bestatigt werden, heißt das: diese Zeilen loschen.

Sie werden feststellen, daß unter den 8 ersten zu loschenden Zeilen auch die Zeile 1 ist, was nichts anderes heißt, als daß sich der VC-20 bei der darauf folgenden "Loschaktion" die Nummern der nachsten zu loschenden Zeilen aus der DATA-Zeile 2 ausliest, etc. etc. Das Resultat des eben beschriebenen Verfahrens sehen Sie in Bild 6.1.2.

Zeile 170:
Der gesuchte Begriff wird der Variablen S$ übergeben.

Zeile 180: Die Länge S des gesuchten Begriffs wird geprüft.

Zeile 190: PGM beenden?

Zeile 200:
Die Länge S des gesuchten Begriffs wird in die Zero-Page-Adresse 0 geschrieben.

Zeile 220:
Der Suchbegriff wird in das Übergaberegister gePOKEd (Zeile 100) und es folgt der Sprung in das M-PGM SUCHEN.

Zeile 230:
Das M-PGM hat den DATA-Vektor auf das Datum gerichtet, das den gesuchten Begriff enthält. Wenn der Begriff nicht gefunden wird, enthalten die DATA-Vektoren (#63,64 und #65,66; siehe Anhang A.1) die Adresse des DATA-ENDE-ZEICHENs (@). Das gefundene Datum wird gelesen und der Variablen SS$ übergeben.

Zeile 240:
SS$ wird auf dem Bildschirm ausgegeben. Es folgt der Sprung in das UP für die Ausgabe von "Weiter mit Taste". Falls nicht "Pfeil nach oben" eingegeben wird, geht es weiter mit dem Sprung in das M-PGM WEITERSUCHEN. Das gefundene Datum wird wiederum auf dem Bildschirm dargestellt.

Zeile 250:
Der gesuchte Begriff wurde nicht gefunden. Das PGM wird an Zeile 170 fortgesetzt.

Zeile 260:
UP für die Ausgabe "Weiter mit Taste"; falls man "Pfeil nach oben" eingibt, wird die augenblickliche DATA-Zeilennummer angezeigt und das PGM verlassen.

```
100 REM...16 STELLEN...*
120 A=(PEEK(46)-1)*256+PEEK(45):B=A+153
170 S$="":PRINTCHR$(147):PRINT:PRINT:PRINT:INPUT"SUCHBEGRIFF";S$
180 S=LEN(S$):IFS<1ORS>16THEN120
190 IFS$="↑"THENEND
200 POKE0,S
220 FORI=1TOS:POKEPEEK(44)*256+PEEK(43)+4+I,ASC(MID$(S$,I,1)):NEXT:SYSA
230 READSS$:IFSS$="@"THEN250
240 PRINT:PRINT:PRINTSS$:GOSUB260:SYSB:GOTO230
250 PRINT:PRINT:PRINT"NICHT (MEHR) GEFUNDEN":GOSUB260:GOTO170
260 PRINT:PRINTCHR$(29)CHR$(18)"WEITER MIT TASTE"CHR$(146)
270 GETE$:IFE$=""THEN270
280 IFE$<>"↑"THENRETURN
290 PRINTCHR$(147)CHR$(145)"L↖"PEEK(63)+256*PEEK(64)
300 POKE631,19:POKE632,13:POKE198,2:END
60000 DATA@
```

Bild 6.1.2

7b) Erläuterungen zum M-PGM (s. Bild 6.1.3):

Teil 1 ($2000 - $2098):
Dieser Teil des M-PGMs durchsucht die DATA-Zeilen nach dem vorgegebenen Begriff (SUCHEN). Der DATA-Vektor wird so eingestellt, daß das gesuchte Datum mit READ gelesen und einer Variablen zugewiesen werden kann.

Teil 2 ($2099 - $20F6):
stellt die PGM-Vektoren so ein, daß in Teil 1 des PGMs hinter dem zuletzt gelesenem Datum nach dem vorgegebenen Begriff gesucht werden kann (WEITERSUCHEN).

```
2000 LDA $2B                    TEIL 1
2002 STA $03                    1. Basic-Zeile
2004 LDA $2C
2006 STA $04
2008 LDY ##$00
200A LDA ($2B),Y                2. Basic-Zeile
200C STA $FB
200E INY
200F LDA ($2B),Y
2011 STA $FC
2013 NOP
2014 LDA $03                        NAECHSTE ZEILE
2016 STA $01                    vorige Zeile
2018 LDA $04
201A STA $02
201C LDA $FB                    aktuelle Zeile
201E STA $03
2020 LDA $FC
2022 STA $04
2024 LDY ##$00
2026 LDA ($03),Y                naechste Zeile
2028 STA $FB
202A INY
202B LDA ($03),Y
202D STA $FC
202F INY
2030 LDA ($03),Y                Zeilennummer
2032 STA $3F
2034 INY
2035 LDA ($03),Y
2037 STA $40
2039 NOP
203A INY                            DATA-ZEILE SUCHEN
203B LDA ($03),Y
203D CMP ##$83                  DATA-Zeile ?
203F BNE $2014
2041 DEY
2042 STY $05
2044 NOP
2045 INC $05                        BEGRIFF SUCHEN
2047 LDY $05
2049 LDX ##$00
204B NOP
204C INY                            NAECHSTES ZEICHEN
204D LDA ($03),Y                Zeilenende ?
204F BEQ $2014
2051 CMP ##$40                  DATA-Ende ?
2053 BEQ $2060
2055 CMP $1206,X                vergleichen
2058 BNE $2045
205A INX
205B CPX $00                    Begriff gefunden ?
205D BNE $204C
205F NOP
2060 DEY                            DATA-VEKTOR SETZEN
2061 LDA ($03),Y
2063 CMP ##$2C                  Komma ?
2065 BEQ $2086
```

Bild 6.1.3 (Teil 1)

```
2067 CMP ##83                    DATA ?
2069 BNE $2060
206B NOP
-------------------------------------------------------------------
206C LDY ##02                        ZEIGER NACH VORIGE ZEILE
206E LDA ($01),Y                 Zeilennummer
2070 STA $3F
2072 INY
2073 LDA ($01),Y
2075 STA $40
2077 NOP
-------------------------------------------------------------------
2078 INY                             ZEILENENDE SUCHEN
2079 LDA ($01),Y                 Zeilenende ?
207B BNE $2078
207D LDA $01
207F LDX $02
2081 STA $03
2083 STX $04
2085 NOP
-------------------------------------------------------------------
2086 LDX $04                         ZEIGER NACH AKTUELLE ZEILE
2088 LDA $03                     setzt DATA-Zeiger
208A STY $05                     auf Komma oder
208C CLC
208D ADC $05                     Zeilenende (0) vor
208F BCC $2092
2091 INX                         dem gesuchten Begriff
-------------------------------------------------------------------
2092 STX $42                         ZEIGER SETZEN
2094 STA $41
2096 RTS
-------------------------------------------------------------------
TEIL 2:      aehnlich wie 1. Teil

2097 BNE $2045                   20C8 INY
2099 LDA $2B                     20C9 LDA ($03),Y
209B STA $03                     20CB CMP $3F
209D LDA $2C                     20CD BNE $20AD
209F STA $04                     20CF INY
20A1 NOP                         20D0 LDA ($03),Y
20A2 LDY ##00                    20D2 CMP $40
20A4 LDA ($03),Y                 20D4 BNE $20AD
20A6 STA $FB                     20D6 LDA $04
20A8 INY                         20D8 CMP $42
20A9 LDA ($03),Y                 20DA BEQ $20EA
20AB STA $FC                     20DC CLC
20AD LDA $03                     20DD LDA ##00
20AF STA $01                     20DF SBC $03
20B1 LDA $04                     20E1 CLC
20B3 STA $02                     20E2 ADC $41
20B5 LDA $FB                     20E4 STA $05
20B7 STA $03                     20E6 DEC $05
20B9 LDA $FC                     20E8 BNE $2097
20BB STA $04                     20EA CLC
20BD LDY ##00                    20EB LDA $41
20BF LDA ($03),Y                 20ED SBC $03
20C1 STA $FB                     20EF STA $05
20C3 INY                         20F1 DEC $05
20C4 LDA ($03),Y                 20F3 BNE $2097
20C6 STA $FC                     20F5 BEQ $2097
```

Bild 6.1.3 (Teil 2)

6.2 SORTIEREN

Eine kleine Sortierroutine darf in einer PGM-Sammlung naturlich nicht fehlen. Wie fast alle in diesem Buch beschriebenen Programme ist auch diese Sortierroutine universell einsetzbar und ermoglicht dem Anwender eine nur durch den Ausbau des Arbeitsspeichers begrenzte Anzahl von Daten zu sortieren. Gerade weil diese Routine so universell ist, mußten einige Kompromisse eingegangen werden. So ist die in diesem Kapitel vorgestellte Sortierroutine nicht die schnellste, aber dafur sehr zuverlassig. Wer selbst schon einmal ein Sortier-PGM geschrieben hat, kennt die Probleme, die immer dann auftreten, wenn es darum geht, die Dauer des Sortiervorgangs in vertretbaren Grenzen zu halten. Sollte Ihnen das hier vorgestellte PGM zu langsam erscheinen, so untersuchen Sie doch auch einmal ein anderes Soriterverfahren. Literatur uber Sortierprogramme gibt es reichlich und man sagt: An der Qualität eines Sortier-PGMs kann man nicht nur die Leistungsfähigkeit eines Betriebssystems ablesen, sondern auch die des Programmierers.

BEDIENUNGSHINWEISE

Das kleine PGM-Beispiel nach Bild 6.2.1 liest die zu sortierenden Daten aus DATA-Zeilen und schreibt diese in die vorher dimensionierte Variable S$(I). Danach folgt der Sprung in die Sortierroutine ab Zeile 3000. Nach Rückkehr aus der Sortierroutine werden die Daten in sortierter Reihenfolge auf dem Bildschirm ausgegeben. Sie brauchen in Ihrem PGM naturlich nicht jedes Datum nach S$(I) zu schreiben, sondern vielleicht nur jedes 3. oder 4. Datum, um eine Adressdatei nach Postleitzahlen oder Ortsnamen zu sortieren oder ähnliches. Die Daten mussen nicht in DATA-Zeilen stehen, Sie konnen diese auch aus einer Datei (auf Kassette oder Diskette) lesen und der Variablen S$(I) zuweisen. Das Sortieren in umgekehrter alphabetischer Reihenfolge ist naturlich auch moglich, dazu muß in Zeile 3000 das Zeichen > (größer) in < (kleiner) geändert werden. Aber Vorsicht, das Zeichen = (gleich) darf nicht fortgelassen werden, andernfalls würde sich das PGM bei gleichen Daten in einer Endlosschleife festfahren.

PROGRAMM-EINZELHEITEN

1.) Name: SORTIEREN

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Basic-Hilfsroutine

4.) Anzahl der Bytes des UP's: 124 (Zeilen 3000 - 3070)

5.) Benutzte Variablen:

S$	Wird nur benutzt, um DATA-Zeilen auszu lesen und die Anzahl der zu sortierenden Daten zu ermitteln
N	Anzahl der zu sortierenden Daten
R(I)	Reihenfolge der sortierten Daten
S$(I)	Daten in unsortierter Reihenfolge
I	Schleifenzahler
H	Hilfsvariable =1 : es mußte sortiert werden =0 : es mußte nicht sortiert werden

6.) Listing: siehe Bild 6.2.1

7.) Erlauterungen zum Programm:

Zeilen 10 - 40:
Im 1. Teil des PGMs wird die Anzahl der zu sortierenden Daten ermittelt und der Variablen N ubergeben. Die RESTORE-Anweisung setzt den DATA-Vektor zuruck.

Zeile 50:
Die Anzahl der zu sortierenden Elemente wird dimensioniert.

Zeile 60:
Die zu sortierenden Daten werden der Variablen S$(I) zugewiesen. Die Variable R(I) ist der Index fur die unsortierte Reihenfolge der Daten. Nach dem Sortiervorgang enthalt R(I) die sortierte Reihenfolge der Daten.

Zeile 70: Sprung in die Sortierroutine

```
10 REM SORTIEREN
20 READS$
30 IFS$<>"@"THENN=N+1:GOTO20
40 RESTORE
50 DIMS$(N),R(N)
60 FORI=1TO N:R(I)=I:READS$(R(I)):NEXT
70 GOSUB3000:PRINT"J"
80 FORI=1TO N
90 PRINT:PRINTS$(R(I))
100 NEXT
110 END
3000 H=0
3010 FORI=1TON
3020 IFS$(R(I))=>S$(R(I-1))THEN3050
3030 R(0)=R(I):R(I)=R(I-1):R(I-1)=R(0)
3040 H=1
3050 NEXT
3060 IFH=1THEN3000
3070 RETURN
10000 DATA SCHULZ H.,MUELLER,MEIER,SCHAEFER,SCHULZ A.,PETERS,MAYER
60000 DATA @
```

Bild 6.2.1

Zeilen 80 - 100:
Die Daten werden in sortierter Reihenfolge auf dem Bildschirm ausgegeben.

Zeile 3000:
Die Hilfsvariable H wird 1, wenn sortiert wurde.

Zeile 3010:
In dieser FOR-NEXT-Schleife werden die zu sortierenden Daten miteinander verglichen. Es wird immer das 2. mit dem 1. Element, das 3. mit dem 2. Element usw. bis zum N. Element verglichen.

Zeile 3020:
Wenn z.B. das 2. Element größer als das 1. Element (I-1) ist, dann sind die beiden Elemente schon in der richtigen Reihenfolge angeordnet und es erfolgt der Sprung in Zeile 3050. Wenn nicht, dann werden die Zeilen 3030 und 3040 abgearbeitet.

Zeile 3030:
Der eigentliche Sortiervorgang besteht darin, daß der Index fur die Reihenfolge der beiden soeben verglichenen Elemente vertauscht wird. R(0) dient dabei nur als Zwischenspeicher fur R(I). Dieses Sortierverfahren nennt man Bubble-Sort.

Zeile 3040:
Die Hilfsvariable H wird "gesetzt" und ist ein Flag dafür, daß sortiert werden mußte.

Zeile 3050:
NEXT: die nächsten beiden Elemente miteinander vergleichen

Zeile 3060:
Die Zeilen 3000 bis 3050 müssen so lange durchlaufen werden, bis H nicht mehr gesetzt wird. Erst dann ist der Sortiervorgang beendet und R(I) enthalt jetzt die sortierte Reihenfolge der Daten.

Zeile 3070:
Rücksprung in das Hauptprogramm

Zeilen 10000 - ...:
Unsortierte Daten in DATA-Zeilen. Das letzte Datum muß "@" sein, oder das PGM steigt mit der Fehlermeldung "? OUT OF DATA ERROR" aus.

6.3 LÖSUNG DES PORTABILITÄTS-PROBLEMS

Es mußte leider einen popularen Heimcomputer, wie es ja der VC-20 ohne Zweifel ist, treffen: sein Benutzer legt sich anfangs, wenn er sich mit der Grundversion zufrieden gibt, alle moglichen schönen Programme zu und ist plötzlich arg enttäuscht oder verblüfft (je nach Stimmungslage), wenn diese ja so gut funktionierenden Programme nur noch Mist verursachen, nachdem eine Speichererweiterung in den VC-20 hineingesteckt worden ist.

Diese Tatsache ist schon so lange bekannt, wie es den VC-20 auf dem Markt gibt. Leider hat der Hersteller in keinster Weise fur

eine fur den Benutzer einfach geloste Umschaltemoglichkeit gesorgt, noch druckt er ihm von Anfang an ausreichende Informationen in die Hand.

Dieser Effekt, daß namlich VC-20-Programme auf dem einen Rechner laufen und auf dem anderen nicht, ist als Portabilitäts-Problem allgemein bekannt geworden. Er tritt deswegen auf, weil der Adreßraum fur Video- und Farbspeicher (s. Anhang A.2) für unterschiedliche Speicherausbaustufen ebenfalls anders ist. Programme, die mithilfe PEEK- und POKE-Befehlen, oder auch per Maschinensprache, raffiniertere Moglichkeiten aus dem VC-20 herausholen wollen, konnen verstandlicherweise solche Anderungen von sich aus und von selbst nicht mitvollziehen. Abhilfe schafft hierbei naturlich die "Hau-Ruck"-Methode, indem man einfach die fur das jeweilige Programm notwendige Speichererweiterung hineinsteckt bzw. herauszieht oder ein- bzw. ausschaltet. Letzteres ist noch gerade zumutbar, ersteres jedoch recht umstandlich und erhoht den Verschleiß fur sowohl den Expansion-Port als auch die Speichererweiterung.

Eine weitere Moglichkeit, den VC-20 in die fur die jeweilig notwendige Speicherausbaustufe "umzuschalten" ist die softwaremaßige. Bezuglich des Portabilitäts-Problems reicht es aus, 3 Typen von Speicherausbaustufen zu unterscheiden:

- Grundversion (GV)
- 3K-Version (3K-V), also ein um einen 3 KByte-Zusatzspeicher (Adreßraum $0400-0FFF) erweiterten VC-20
- Ausbauversion (AV), mit einem Zusatzspeicher von >= 8 KByte

1A) UMSCHALTUNG: AV --> GV oder 3K-V ---> GV

In diesem Fall geben Sie im Direkt-Modus hintereinander ein:

```
POKE 642,16:POKE 781,0: SYS 64970:SYS 64818 (Return)
```

Auf dem Bildschirm wird das gewohnte Einstiegsbild angezeigt, allerdings mit der Meldung: "3583 BYTES FREE". Dem VC-20 haben wir also somit vorgegaukelt, daß er nicht die geringste Speichererweiterung aufweist. Diese ist naturlich nach wie vor uber POKE- und PEEK-Befehle erreichbar, auch konnen wie gewohnt dort M-PGMe abgelegt und zusatzlich benutzt werden.

2A) UMSCHALTUNG: AV -> 3K-V

Dies ist nur sinnvoll, wenn tatsachlich im Adreßbereich $0400-0FFF RAM vorhanden ist. Folgendes wird eingegeben:

POKE 781,0:POKE 782,4:SYS 65162:SYS 64970:SYS 64818 (Return)

Das gewohnte Einstiegsbild wird wieder auf den Bildschirm gebracht, hier mit der Meldung: "6655 BYTES FREE". Dies geschieht auch dann, wenn Sie in Wirklichkeit keinen 3 KByte-Zusatzspeicher an den VC-20 angeschlossen haben.

3A) UMSCHALTUNG: GV --> AV oder 3K-V --> AV

Sie haben richtig gelesen, es ist kein Irrtum. Im ersten Moment stutzt man naturlich und fragt sich, ob es denn uber eine bloße Umschaltung moglich ist, ein nicht ausgebautes in ein ausgebautes Gerät umzuwandeln. Das wird selbstverstandlich in keinster Weise gemacht. Nur, manche Programme laufen halt nur dann, wenn der Video-Speicher im Bereich $1000-1FF9 und der Farbspeicher im Bereich $9400-95F9 liegt. Diese Verteilung entspricht derjenigen der AV. Eine Umschaltung ist also auch dann erforderlich, wenn das Programm zwar nur höchstens 3583 Bytes umfassen darf, jedoch die Speicherverteilung gemaß AV benötigt. Hierzu ist folgende Eingabe erforderlich:

POKE 781,0:POKE 782,32:SYS 64978:SYS 64818 (Return)

Die 3 oben besprochenen Umschaltungen sind verhältnismäßig unaufwendig und können schnell vor dem Programm eingegeben werden, worauf sich die entsprechende Umschaltung bezieht. Auch können Sie mit einer Zeilennummer versehen werden und als eigenständiges Programm gewissermaßen als Vorspann-Programm vor dem eigentlichen Programm auf Kassette stehen.

Manche Programmierer wollen jedoch solche Umschaltungen nicht über Basic, sondern per Aufruf entsprechender M-PGMe vornehmen, die darüberhinaus koppelbar mit selbst definierten Befehlen (s. Kap. 4.1) sein sollen. Man stelle sich nur vor, wie luxurios es wäre, wenn Sie beispielsweise den Befehl "GV" im Direkt-Modus eingeben und hernach das VC-20-Einstiegsbild mit "3583 BYTES FREE" erhalten würden.

1B) UMSCHALTUNG: AV ---> GV oder 3K-V ---> GV

```
10 REM GV
20 REM SCHALTET DEN VC20 IN DIE GRUNDVERSION UM
30 L=17
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):AD=AD+SD:GOSUB250:A=AL:GOTO210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:END
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(146):PRINT:RETURN
1000 DATA120,162,0,142,129,2,169,16,141,130,2,32,202,253,76,50,253
```

Bild 6.3.1a

Die Losung hierfur bietet das kurze M-PGM gemaß Bild 6.3.1b. Es ist frei verschiebbar (relocatable) und kann zum Beispiel uber den speziellen Basic-Loader (s. Bild 6.3.1a) aus Kap. 5.2 uberallhin geladen werden. Falls Sie sich für das RAM-Ende als Standort dieses M-PGMs entscheiden, werden bereits existierende M-PGMe nicht angetastet und es wird außerdem der Basic-RAM-Ende-Vektor neu ausgerichtet.

```
SEI
LDX #$00
STX $0281
LDA #$10
STA $0282
JSR $FDCA
JMP $FD32
```

Bild 6.3.1b

2B) UMSCHALTUNG: AV -> 3K-V

Es gelten sinngemäß die gleichen Ausführungen wie unter 1B. Die analogen Listings entnehmen Sie den Bildern 6.3.2a sowie 6.3.2b.

```
10 REM 3K-V
20 REM SCHALTET DEN VC20 IN DIE 3K-BYTE-VERSION UM
30 L=14
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):AD=AD+SD:GOSUB250:A=AL:GOTO210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:END
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SDCHR$(146):PRINT:RETURN
1000 DATA120,162,0,160,4,32,138,254,32,202,253,76,50,253
```

Bild 6.3.2a

```
SEI
LDX #$00
LDY #$04
JSR $FE8A
JSR $FDCA
JMP $FD32
```

Bild 6.3.2b

3B) UMSCHALTUNG: GV --> AV oder 3K-V --> AV

Es gelten sinngemäß die gleichen Ausführungen wie unter 1B. Die analogen Listings entnehmen Sie den Bildern 6.3.3a sowie 6.3.3b.

```
10 REM GV (8K-V)
20 REM SCHALTET DEN VC20 IN DIE >=8K-BYTE-VERSION UM
30 L=11
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):AD=AD+SD:GOSUB250:A=AL:GOTO210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:END
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SIDCHR$(146):PRINT:RETURN
1000 DATA120,162,0,160,32,32,210,253,76,50,253
```

Bild 6.3.3a

```
SEI
LDX #$00
LDY #$20
JSR $FDD2
JMP $FD32
```

Bild 6.3.3b

6.4 SEITENSPRUNG

Die Tabelle A.5.6 in Anhang A.5 gibt darüber Aufschluß, wie das "Fenster", durch das man in den VC-20 hineinschaut, in 8 möglichen Fällen "versetzt" werden kann. Da jedes dieser Fenster einen der 8 VC-20-Speicherbereiche von je 506 Bytes (genau genommen sind es 512, die letzten 6 jedoch sind nicht von vornherein sichtbar) durch den Bildschirm hindurch dem Benutzer visuell zugänglich macht, stößt man auf die Frage, ob es denn nicht moglich ist, mit "umschaltbarem" Bildschirm alle diese 8 Speicherbereiche zur gleichen Zeit zu benutzen.

"Seitensprung" haben wir dieses Umschalten des Bildschirms auf jeden beliebigen dieser 8 Speicherbereiche genannt. Per Druck auf die Funktionstaste werden die Video-Speicherbereiche gemäß Bild 6.4.1 sichtbar gemacht. Allerdings muß hierbei vorausgesetzt werden, daß Sie mindestens eine 3KByte- oder eine 8KByte-Speichererweiterung an Ihren VC-20 angeschlossen haben.

F1: $1000-11F9	F3: $1400-15F9	F5: $1800-19F9	F7: $1C00-1DF9
F2: $1200-13F9	F4: $1600-17F9	F6: $1A00-1BF9	F8: $1E00-1FF9

Bild 6.4.1

Da von vornherein jedoch der Basic-Anfang in der VC-20-Ausbauversion (>=8KByte Erweiterung) bei $1200 liegt, muß der Basic-Arbeitsspeicherbereich außerhalb aller in Bild 6.4.1 aufgeführten Bereiche installiert werden. Wie dies gemacht wird, haben wir im Kapitel 2.1 gelernt. Wenn wir den Basic-Arbeitsspeicher unmittelbar auf die 8 Seiten-Speicher folgen lassen wollen, also ab $2000=#8192, geben wir im Direkt-Modus ein:

```
POKE 44,32:POKE 8192,0:NEW    (Return)
```

Beim VC-20 in der 3K-Version fängt der Basic-Arbeitsspeicher zwar schon bei $0400 an, ragt jedoch in die Seiten-Speicherbereiche hinein, so daß er oben "abgeschnitten" werden muß mit:

```
POKE 56,16:CLR    (Return)
```

BEDIENUNGSHINWEISE

Das Programm SEITENSPRUNG 1 (s. Bild 6.4.2) wird nach der Verschiebung des Arbeitsspeichers normal mit LOAD geladen und hernach mit RUN gestartet. Nach einer kurzen Pause tut sich einiges auf dem Bildschirm. Sie können beobachten, daß nacheinander die Seiten-Speicher sämtlich mit den Zahlen 1 bis 8 gefüllt werden. Wenn dieser Prozeß abgeschlossen ist, betätigen Sie die Funktionstasten und sehen anhand der Zahlen auf dem Bildschirm, in welcher Seite Sie sich gerade befinden.

```
0 REM SEITENSPRUNG 1
10 FORI=0TO1023:POKE37888+I,6:NEXT
20 FORBP=0TO7
30 GOSUB1000
40 FORI=0TO505:POKEAD+1,49+BP:NEXT
50 NEXT
60 WAIT198,1:GETE$
70 H=ASC(E$)-133:IFH<0ORH>7THEN60
80 BP=H*2-(HAND4)/4*7
90 GOSUB1000:GOTO60
1000 POKE648,2*BP+16:AD=PEEK(648)*256
1010 POKE36866,(PEEK(36866)AND127)+((BPAND1)*128)
1020 POKE36869,(PEEK(36869)AND15)OR((BPAND6)*8+192)
1030 RETURN
```

Bild 6.4.2

Naturlich hat dieses Programm mehr Demonstrationscharakter als Nutzwert. Anders sieht es jedoch bei SEITENSPRUNG 2 aus (s. Bild 6.4.3).

```
0 REM SEITENSPRUNG 2
10 DIMLI%(7,23)
20 FORBP=0TO7
30 GOSUB1000
40 PRINTCHR$(147)"VIDEO-SPEICHER";BP+1
50 GOSUB3000
60 NEXT:BP=7
70 GOSUB2000
80 H=ASC(E$)-133:IFH<0ORH>7THEN70
90 GOSUB3000
100 BP=H*2-(HAND4)/4*7
110 GOSUB3010
120 GOSUB1000:GOTO70
1000 POKE648,2*BP+16:POKE210,PEEK(648)
1010 POKE36866,(PEEK(36866)AND127)+((BPAND1)*128)
1020 POKE36869,(PEEK(36869)AND15)OR((BPAND6)*8+192)
1030 PRINTCHR$(19):RETURN
2000 GETE$:IFE$=""THEN2000
2010 PRINTE$;:IFPEEK(214)=22THENPOKE214,18
2020 RETURN
3000 FORI=0TO23:LI%(BP,I)=PEEK(217+I):NEXT:RETURN
3010 FORI=0TO23:POKE217+I,LI%(BP,I):NEXT:RETURN
```

Bild 6.4.3

Nachdem Sie dieses in der gleichen Weise wie letzteres zum Laufen gebracht haben, sehen Sie anstatt lauter Zahlen nur das Wort "VIDEO-SPEICHER" und die Seitennummer in der 1. Zeile des Bildschirms. Wieder konnen Sie per Betatigung der Funktionstasten beliebig zwischen den Video-Speichern hin und her schalten. Zusatzlich ist es jedoch möglich, die einzelnen Seiten nach Belieben vollzuschreiben.

Mit SEITENSPRUNG 2 erhalten Sie somit ein Werkzeug, welches Sie in denjenigen Programmen einsetzen konnen, bei denen Sie simultan bis zu 8 verschiedene "Seiten" bearbeiten wollen.

PROGRAMM-EINZELHEITEN

1.) Name: SEITENSPRUNG 1 und 2

2.) Ausbaustufe: alle Versionen außer GV

3.) Art des PGMs: Hilfsprogramm

4.) Anzahl der Bytes
SEITENSPRUNG 1 : 300
SEITENSPRUNG 2 : 451

5.) Benutzte Variablen:

BP	Bildschirm-Page (0...7)
AD	Anfangsadresse des momentanen Video-Speichers
H	ASCII-Code der Funktionstaste abzgl. 133
E$	Eingegebene Funktionstaste
I	Schleifenzähler

6.) Listings
SEITENSPRUNG 1 : Bild 6.4.2
SEITENSPRUNG 2 : Bild 6.4.3

7a) Erläuterungen zu SEITENSPRUNG 1:

Zeilen 10 - 50: Initialisierung
Sämtliche Farbspeicherzellen bekommen die Farbe "BLAU" (Inhalt=6) zugewiesen, sonst bleiben nämlich die meisten Zahlen

unsichtbar. In den Zeilen 20 und 30 wird nacheinander jeder der Video-Speicher eingeschaltet. In Zeile 40 wird in jede Speicherzelle des Video-Speichers die Seitennummer hineinge-POKEd.

Zeilen 60 - 90: Funktionstastenabfrage
In Zeile 60 wird ermittelt, daß eine Taste gedruckt wurde. In Zeile 70 wird gepruft, ob es sich dabei wirklich um eine Funktionstaste handelt. Die Formel in Zeile 80 rechnet den um 133 verminderten ASCII-Code der Funktionstaste (Variable H) in die korrelierende Bildschirm-Page-Nummer um.

Zeilen 1000 - 1030: Video-Speicher-Umschaltung
Die hier aufgefuhrten Anweisungen basieren auf den in Tabelle A.5.6 zusammengestellten Werten fur die Speicherzellen #648, #36866 sowie #36869.

7b) Erlauterungen zu SEITENSPRUNG 2:

Die Ahnlichkeit zu SEITENSPRUNG 2 werden Sie bereits erkannt haben. Folgendes kommt noch hinzu:

Bevor in eine andere Bildschirm-Page umgeschaltet wird, wird die in den Zellen #217-240 gespeicherte Zeilenverkettungs-Tabelle in das Feld LI gerettet (UP 3000) und vor Beschreiben des SeitenSpeichers aus LI ausgelesen (UP 3010).

In Zeile 2010 wird das Bildschirm-Scrolling dadurch verhindert, daß die Cursor-Zeilenposition (Zelle #214) um 4 Zeilen nach oben hin verändert wird (denn eine logisch zusammenhangende Zeile kann aus 4 Bildschirmzeilen bestehen).

6.5 AUTOSTART FÜR BASIC- UND MODULPROGRAMME

Viele von Ihnen werden sicherlich schon eine Reihe von den VC-20-Modulspielen - so heißen diese ja wohl im VC-20-Programmierer-Jargon - besitzen. Ob Sie diese nun als "Cartridge" zum Hineinstecken in den Expansion-Port oder aber als Kassette, Diskette etc. präsent haben, sie alle haben eine Eigenschaft gemeinsam: sie starten sich nach Einschalten bzw. Drücken der RESET-Taste

(s. Anhang A.2) bzw. Eingabe von SYS 64802 alle von selbst. Warum das so ist, wird im 2. Teil dieses Kapitels besprochen.

1. AUTOSTART FÜR BASIC-PROGRAMME

Es wird sich allerdings so mancher Leser gefragt haben: "Gibt's denn so etwas auch für Basic-Programme?" Die Antwort kann mit "Ja" beantwortet werden. Das im folgenden vorgestellte Programm "BASIC-AUTOSTART" macht dies möglich, indem es einen "präparierten" Vorspann erzeugt, der vor das eigentliche Programm abgespeichert werden muß, welches Sie automatisch starten lassen wollen.

BEDIENUNGSHINWEISE

Nachdem "BASIC-AUTOSTART" mittels der üblichen Lademethoden geladen wurde, wird es mit RUN gestartet. Ziel ist es jetzt, ein Vorprogramm zu dem eigentlichen später abzuSAVEnden PGM zu erzeugen und auf Band abzuspeichern. Dies geschieht so:

a) Erst wird hintereinander eingegeben:
POKE 44,3:POKE 45,94:POKE 46,3:CLR:A=4614 (Return)

Der Wert A=4614 gilt für die AV (sämtliche Speicherausbauversionen >= 8 KByte). Fur GV gilt A=4102 und für 3K-V gilt A=1030.

b) Dann wird die Schleife eingegeben:
FOR I=0 TO 12:A$=A$+CHR$(PEEK(A+I)):NEXT (Return)

c) Zum Abschluß (allerdings erst das Band auf gewunschte Position bringen):
POKE 770,81:POKE 771,3:SAVE"(ein aus genau 16 Stellen bestehender Name)"+A$,1,1 (Return)
Es wird jetzt "PRESS RECORD & PLAY ON TAPE" erscheinen und Sie befolgen diese Anweisung, verfolgen das weitere Geschehen auf dem Bildschirm. Denn sobald danach:

READY.
LOAD
SEARCHING

erscheint, schalten Sie sofort die Datassette ab.

d) Nachdem Sie per AUS/EINschalten oder Drucken auf RESET alles in den Ausgangszustand versetzt haben, laden Sie, wie gewohnt, Ihr eingentliches Basic-PGM, das Sie mittels dem eben generierten Vorprogramm beim zukünftigen Laden aller beiden Programme automatisch starten lassen möchten.

```
10 REMXXXXXXXXXXXXX
20 REM IN ZEILE 10 BEFINDET SICH DAS AUTOSTART-MPGM
30 FORI=0TO12:READA:POKEPEEK(43)+256*PEEK(44)+5+I,A:NEXT
40 PRINTCHR$(147)"DIESES PGM DIENT ZUR  ERZEUGUNG EINES AUTO- START-VORPROGRAMMS
50 PRINT:PRINT"DIESES VORPROGRAMM ISTUNIVERSELL FUER BASIC-PGM AUF DATASSETTE"SP
C(4)
60 PRINT"EINSETZBAR.":PRINT:PRINT:PRINTCHR$(18)"TASTE"
70 GETA$:IFA$=""THEN70
80 PRINTCHR$(147)"HIER DIE "CHR$(18)"SAVE-REGEL"CHR$(146)":"
90 PRINT:PRINT"EINGEBEN: POKE44,3:POKE45,94:POKE46,3:CLR:A=4614 "CHR$(18)"RETURN
";
100 PRINTCHR$(146)"  (FUER 6V:A=4102; 3K-V: A=1030)"    "
110 PRINT:PRINT"FOR I=0TO12:A$=A$+CHR$(PEEK(A+I)):NEXT "CHR$(18)"RETURN"
120 PRINT"POKE770,31:POKE771,3:SAVE"CHR$(34)"GENAU 16 STELLEN";
130 PRINTCHR$(34)"+A$,1,1 "CHR$(18)"RETURN"
140 PRINT:PRINT"WENN DANACH "CHR$(18)"SEARCHING"CHR$(146)" ERSCHEINT:";
150 PRINT"REKORDER AB-SCHALTEN.";
160 PRINT" RESET-TASTE DRUECKEN BZW. GERAET  ABSCHALTEN.";
170 PRINT" NEUES PGM LADEN UND NORM. SAVEN"
180 DATA32,91,228,169,131,141,119,2,230,198,76,131,196
```

Bild 6.5.1

Warum funktioniert dies nun?

In a) wird der Basic-Anfang-Vektor (Adressen $002B,002C = #43,44) auf die Speicherzelle $0301 = #769 "verbogen", denn wenn die Zelle #44 den Wert 3 erhält, die Zelle #43 ohnehin schon 1 enthält, so ergibt sich als Zieladresse dieses Vektors: 3*256+1 = #769. Außerdem wird der Variablen-Anfang-Vektor (Adressen $002D,002E = #45,46) auf die Speicherzelle $035E = #862 gelegt.

Wurde man jetzt die Funktion "SAVE" benutzen, so würden die Inhalte der Zellen #769-861 auf Band gespeichert werden, weil der VC-20 immer "stur" alles zwischen Basic-Anfang (incl. 1. Speicherzelle) und Variablen-Anfang (excl. 1. Speicherzelle) abspeichert.

In b) wird die Stringvariable A$ aus den 13 M-PGM-Bytes der 1. REM-Zeile des Basic-Loaders erzeugt. Testen Sie selbst, daß A$

wirklich aus 13 Bytes besteht, indem Sie PRINT LEN(A$) eingeben und RESTURN drucken.

Die eigentliche Pointe des ganzen in diesem Kapitel beschriebenen Verfahrens - woanders sonst als am Schluß konnte sie auch sein - geschieht schließlich in c):

Die drei in c) stehenden Anweisungen müssen unbedingt hintereinander eingegeben werden. Denn: mit POKE 770,81 und POKE 771,3 wird der Basic-Eingabe-Warteschleife-Vektor verändert und zeigt jetzt auf die Adresse $0351 = #849 (= 81 + 3*256). Wenn nach dieser "Verbiegung" des Vektors RETURN gedrückt würde, gäbe es einen VC-20-Absturz, weil der Vektor nicht mehr zur sonst benutzten Basic-Eingabe-Warteschleife zeigt, sondern ins Nichts (ab $0351 gibt es ja noch kein vernünftiges M-PGM).

Dafür, daß der Vektor $0302,0303 nicht mehr ins Leere zeigt, sorgt der dritte Befehl SAVE"(16 Zeichen)"+A$,1,1. Dieser Befehl macht folgendes:

Zuerst werden die unbedingt genau 16 Zeichen (auch Steuerzeichen wie CLEAR-Taste oder Farbanweisung CTRL- + GRN-Taste gelten als 1 Zeichen) mit dem String A$ verbunden, der aus dem M-PGM besteht. Die aus "(16 Zeichen)" und A$ bestehende String-Kombination wird mittels Befehl SAVE zuerst in den Kassettenpuffer ab $0341 = #833 gelegt und auf Band aufgezeichnet, so daß das einzuspringende M-PGM tatsächlich (zahlen Sie es nach !) ab $0351 = #849 beginnt.

Nicht zu vergessen sind die Kassettenpuffer-Zellen $033C-0340 bzw. #828-832, welche bei Ausführung von SAVE und, wenn alles fertig ist, von LOAD die Werte beinhalten:

Adresse ($):	033C	033D	033E	033F	0340
Inhalt ($):	03	01	03	5E	03

Die Paare $033D,033E sowie $033F,0340 enthalten immer die Kopien der Paare $002B,002C bzw. $002D,020E, also Basic-Anfang und Variablen-Anfang = Basic-Ende abzüglich 1 Byte.

Das Byte in $033C (s. Kap. 3.2 "Tape-Header") jedoch entscheidet, ob der Autostart funktioniert. Es erhält den Wert 03, welcher ebenfalls auf Band aufgezeichnet wird, wenn hinter dem Namen nach

SAVE noch - wie oben gezeigt - zusätzlich ",1,1" eingegeben wird. Wird beim zukünftigen LOAD dieser Wert 03 eingelesen, so heißt das für das VC-20-Betriebssystem: das nach dem Header folgende PGM soll genau in den Bereich geladen werden, der durch die Zeiger $033D-0340 angegeben ist.

Was geschieht also bei LOAD, wenn alles korrekt auf Band aufgezeichnet ist? Das Betriebssystem liest zuerst den Header des vorhin abgeSAVEten PGMs ein, und mit ihm ja auch das M-PGM. Außerdem erhält es mittels 03 in $033C die Information, daß das nach dem Header folgende PGM genau in den Bereich zwischen 0301 und 035D (einschl.) abzuspeichern ist (dabei wird der M-PGM-Speicherbereich sogar nochmals mit gleichem Inhalt "überschrieben"), was dann auch geschieht.

Bild 6.5.2a: Normaler Einsprung in die Basic-Eingabe-Warteschleife

Der Basic-Eingabe-Warteschleife-Vektor wurde aber durch diesen Ladevorgang in der gewünschten Weise geändert, so daß nach Beendigung des Ladevorgangs erst die "Umleitung" (s. Bild 6.5.2a und

2b) über das M-PGM durchlaufen wird. In der "Umleitung" wird der Tastaturpuffer (s. Beschreibung in 7b in den "Programm-Einzelheiten") mit der Tastenkombination SHIFT + RUN/STOP als Reprasentant der Funktion LOAD + anschließendem RUN gefullt. Nach Ruckkehr zur Basic-Eingabe-Warteschleife bleibt dem VC-20 dann wohl nichts anderes mehr übrig, als LOAD + RUN = Autostart auszufuhren.

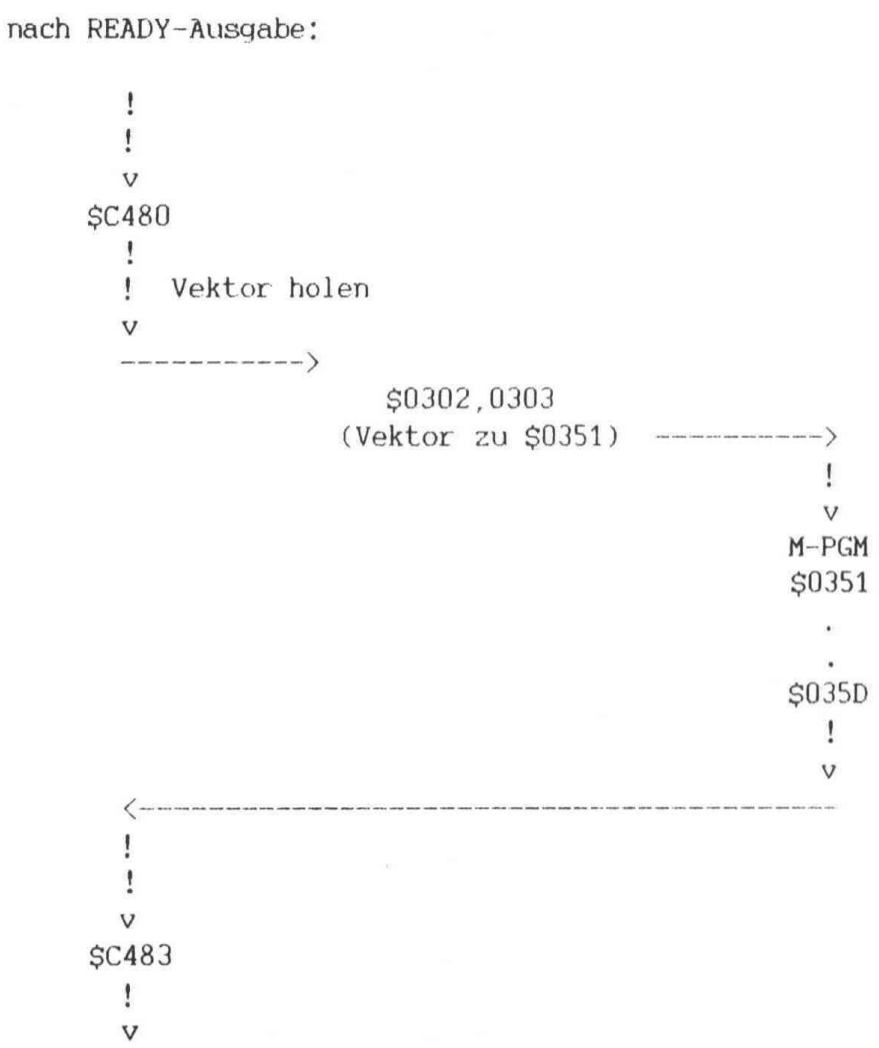


Bild 6.5.2b: Umgeleiteter Einsprung in die Basic-Eingabe-Warteschleife

Angemerkt sei hierbei die wichtigste Tatsaache, daß bei LOAD mit erkanntem Wert 03 für $033C zwar ein PGM genau in den zwischen eingelesenem Basic-Anfang und Variablen-Anfang definierten Speicherbereich abgelegt wird, der Basic-Anfang-Vektor in $002B,002C bzw. #43,44 in keinster Weise verandert wird (der Variablen-Anfang-Vektor allerdings schon, was aber nichts ausmacht). Dies ist auch der Grund dafür, daß das eigentlich einzuLOADene PGM gleich richtig ab normalen Basic-Anfang abgelegt wird (und nicht ab $0301).

Der Zweck eines Autostart-PGMs liegt nicht nur darin, daß man so manche VC-20-Benutzer verblüffen kann, sondern daß sich damit eine weitere Kopierschutzmoglichkeit (s. Kap. 6.8 "Softwareschutz") auftut. Denn ist das PGM erst einmal angelaufen, so gibt es die mannigfaltigsten Möglichkeiten, in diesem PGM es selbst so zu verändern, daß ein abSAVEn nach PGM-Beendigung keinen Sinn mehr ergibt.

PROGRAMM-EINZELHEITEN

1.) Name: BASIC-AUTOSTART

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Basic-Loader

4.) Anzahl der Bytes
   des Basic-Loaders: 876
   des M-PGMs: 13

5.) Benutzte Variablen:

I	Laufvariable
A$	Eingegebenes Zeichen
A	Hilfsvariable

6.) Listings
   des Basic-Loaders: Bild 6.5.1
   des M-PGMs: Bild 6.5.3

```
Zeile
 (1)        JSR $E45B
 (2)        LDA #$83
 (3)        STA $0277
 (4)        INC $C6
 (5)        JMP $C483
```

Bild 6.5.3

7a) Erlauterungen zum Basic-Loader:

Zeilen 10 - 30:
In dieser REM-Zeile wird das eigentliche M-PGM mittels Zeile 30 generiert.

Zeilen 40 - 170:
Ausgabe der Gebrauchsanweisung dieses PGMs

Zeile 180:
Diese DATA-Zeile enthalt die 13 M-PGM-Bytes in DEZ-Code.

7b) Erlauterungen zum M-PGM:

Das M-PGM besteht aus 13 Bytes und wird abgelegt in dem Bereich:

```
$0406-0412 bzw. #1030-1042  für 3K-V
$1006-1012 bzw. #4102-4114  für  GV
$1206-1212 bzw. #4614-4626  für  AV
```

Bei korrekter Befolgung der Gebrauchsanweisung gelangt es fur alle 3 VC-20-Speicherausbauversionen in den Bereich $0351-036D bzw, #849-861 innerhalb des Kassettenpuffers. Zwar konnte es auch ab Adresse $0341=#833 gespeichert werden, wurde aber beim Laden unleserliche Graphik-Zeichen ausgeben. Die Adressen $0341-0350 = 16 Bytes werden sinnvoller für einen auszugebenden Namen ausgenutzt, der beim Laden automatisch zusammen mit "FOUND" ausgegeben wird. Das M-PGM wird ab $0351 gespeichert, weil bei "FOUND" ohnehin nur 16 Bytes des Namens wiedergegeben werden (s. Kopierschutz im Kap. 6.8).

Zeile (1):
Es wird eine Betriebssystemroutine aufgerufen, welche die Basic-Interpreter-Vektoren wie nach Einschalten oder RESET initialisiert. Zwar ist die Wiederherstellung nur vom Vektor in $0302,0303 zur Basic-Eingabe-Warteschleife notig, aber dieses ja schon vorhandene M-UP aufzurufen verbraucht nur 3 Bytes.

Zeilen (2)-(4):
In die 1. Tastaturpufferspeicherzelle wird $83 = #131 gelegt und der Tastaturpufferzahler ($00C6) um 1 erhoht, was dem gleichzeitigen Drücken der SHIFT- und RUN/STOP-Tasten entspricht, d.h. es wird somit dem VC-20 vorgegaukelt, daß der Benutzer diese beiden Tasten gedrückt hat und die Funktion "LOAD" mit darauffolgendem "RUN" wunscht.

Zeile (5):
Jetzt wird in die Basic-Eingabe-Warteschleife gesprungen, die ja regulär sofort angesprungen wird, wenn der Vektor $0302,0303 unverandert bleibt. In dieser Warteschleife wird folgerichtig erkannt, daß im Tastaturpuffer ein Zeichen auf Abarbeitung wartet. Die Folge ist: "SEARCHING", "FOUND ...", "LOADING" etc.

2. AUTOSTART FÜR MODUL-PROGRAMME

Gehen wir nun über zu dem Autostart-Mechanismus von Modul-Programmen, der wahrscheinlich schon vielen bekannt sein dürfte. Was dabei vielleicht weniger geläufig ist, ist die Möglichkeit, sich diesen Mechanismus für eigene Initialisierunswunsche zunutze zu machen.

Was sind eigentlich "Modul-Programme"? Unter diesen Programmen versteht man diejenigen, die zumeist per Steckmodul in den VC-20-Expansion-Port hineingesteckt werden können und sich nach Einschalten des VC-20 von selbst starten. Es sind zumeist Programme, welche innerhalb des A- und B-Bereiches (Adreßraum $A000-BFFF) laufen. Natürlich lassen sich diese Programme auch ohne Hineinstecken eines Moduls laden, wenn im A-B-Bereich RAM vorhanden ist.

Werfen wir einen Blick auf das Prinzip des Autostarts: nach Einschalten des VC-20, Drücken der RESET-Taste oder Eingabe von SYS 64802 wird das VC-20-Betriebssystem aufgefordert, ab Adresse $FD22 seine Arbeit aufzunehmen. In Bild 6.5.4 können wir uns ansehen, was da so am Anfang passiert.

In Zeile FD22 (verwenden wir einfach die Adresse als Zeilennummer, sonst gibt es zu viele Bezeichnungen) wird das X-Register mit dem Wert $FF geladen, in Zeile FD24 wird daraufhin das Interrupt-Diasable-Flag gesetzt (s. Anhang A.2 für mehr Details diesbezüglich), um in diesen anfänglichen Initialisierungsarbeiten nicht "gestört" = unterbrochen zu werden. In Zeile FD25 wird dann der in X stehende Wert $FF in das Stackpointer-Register kopiert, um ein ordnungsgemäßes Funktionieren des Stacks von Anfang an zu gewährleisten. In Zeile FD26 wird der VC-20-Mikroprozessor 6502 auf Byte-Arithmetik "eingestellt".

Adresse (#)	Adresse ($)	
64802	FD22	LDX #$FF
64804	FD24	SEI
64805	FD25	TXS
64806	FD26	CLD
64807	FD27	JSR $FD3F
64810	FD2A	BNE $FD2F
64812	FD2C	JMP ($A000)

Bild 6.5.4

Schließlich gelangen wir zu der Stelle, die für Modul-Programme interessant ist: in Zeile FD27 wird ein Unterprogramm ab FD3F angesprungen. In diesem UP wird geprüft, ob die ASCII-Codes der Zeichen "A0CBM" (letztere 3 Zeichen mit gesetztem Bit 7) ab Adresse $A004 vorliegen (s. Bild 6.5.5).

Ist dies nicht der Fall, geht ohnehin alles seinen gewohnten Gang und der Benutzer wird nach kurzer Zeit das gewohnte VC-20-Einstiegsbild erhalten.

Adresse	Wert	ASCII-Code
A004	41	"A"
A005	30	"0"
A006	C3	"C" mit Bit 7 = 1
A007	C2	"B" mit Bit 7 = 1
A008	CD	"M" mit Bit 7 = 1

Bild 6.5.5

Nehmen wir aber an, daß die gesuchten Werte an der richtigen Stelle gefunden werden. Das UP wird dann mit gesetztem Zero-Flag verlassen, so daß die Anlaufroutine direkt zur Zeile FD2C gelangt. Hier geschieht nun ein indirekter Sprung, den es naher zu erklaren sich lohnt. Der Prozessor liest den Wert aus Adresse $A000 aus und faßt ihn als LOW-Teil der anzuspringenden Adresse auf. Den Wert in $A001 interpretiert er als den HIGH-Teil. Somit ist die Sprungadresse durch den Vektor $A000,A001 bestimmt und dort setzt das Betriebssystem seine Arbeit fort. Steht also z.B. in $A000 der Wert $00 und in $A001 der Wert $B0, dann geht's weiter ab Adresse $B000.

Vom Prinzip nun zur Ausnutzung dieser Eigenschaft: nehmen wir an, Sie wurden gerne den VC-20 dazu veranlassen, einige Dinge wie z.B. CLEAR SCREEN oder Initialisierung von angeschlossenen Geraten etc. automatisch nach Einschalten bearbeiten zu lassen,, so benotigen Sie ein kleines M-PGM z.B. in einem EPROM, welches ab $A000 adreßierbar ist. In $A000,A001 speichern Sie den Sprungvektor, z.B. $09,A0 (entspricht $A009), in $A002,A003 den Sprungvektor fur die Benutzung der Tastenkombination RUN/STOP + RESTORE, namlich $C7,FE (entspricht $FEC7) und in die Adressen $A004-A008 die Werte gemaß Bild 6.5.5.

Das Programm ab $A009 sieht im Prinzip folgendermaßen aus:

```
Adresse ($)
   A009          JSR $FD8D
   A00C          JSR $FD52
   A00F          JSR $FDF9
   A012          JSR $E518
   A015          CLI
   A016          JSR $E45B
   A019          JSR $E3A4
   A01C          JSR $E404
   A01F          LDX #$FB
   A021          TXS
   A022          Initialisierungs-
    .            anweisungen
    .            nach eigenem
    .            Wunsch
    .
    .            JMP $C474
```

Bild 6.5.6

Die Befehle in den Zeilen A009-A021 führen gewissermaßen die noch verbleibenden und auch notwendigen "Restarbeiten" für die VC-20-Initialisierung aus. Ab Zeile A022 trägt man die Ihren Wünschen entsprechenden Befehle ein und schließt die ganze Liste mit dem Befehl JMP $C474 ab; dies ist der Sprung in die Basic-Eingabe-Warteschleife. Stände gleich in Zeile A022 dieser Sprungbefehl, so würde sich der VC-20 in seinem Initialisierungs-Verhalten nicht im geringsten von seinem sonst üblichen Verhalten ohne dieses kleine Modul-Programm unterscheiden.

6.6 VC-20-UHR IN DER 24. ZEILE

Fur diejenigen von Ihnen, verehrte Leser, denen es zu umständlich ist per PRINT TI$ die im VC-20 vorhandene Uhr abzurufen oder die eine ständig sichtbare Uhrzeit während der Abarbeitung eines Programms wünschen, ohne freilich in diesem irgendetwas ändern zu müssen, werden an dem hier besprochenen Programm VC-UHR ihre wahre Freude haben.

Hier wird nämlich die Tatsache ausgenutzt, daß zwischen dem Video-Speicher ($1E00-1FF9 in der GV und 3K-V, $1000-11F9 in der AV) und dem darauf folgendem RAM-Ende (bei der GV und 3K-V) bzw. Basic-Anfang (bei der AV) noch 6 Bytes ungenutzt sind, welche geradezu geschaffen sind, die augenblickliche Uhrzeit anzuzeigen. Da hierfür kein Platz mehr in den bestehenden 23 Zeilen übrig ist, wird einfach zur 24. Zeile übergegangen.

Wird die Uhrzeit per SYS-Befehl "eingeschaltet", so bleibt sie ständig sichtbar, auch während der Abarbeitung beliebiger Programme. Sie läßt sich nach den bekannten Regeln verändern und wird einfach per Drücken der Tasten RUN/STOP und RESTORE wieder "ausgeschaltet".

Das Anzeigen der Uhr erledigt ein Maschinenprogramm, das je nach Ihrem Wunsch in einem beliebigen Speicherbereich generiert wird. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen Überschreiben geschützt und sie können es auch dann aufrufen, wenn Sie beliebige andere Programme im Arbeitsspeicher haben. Weiterhin ist es koppelbar mit von Ihnen frei definierbaren Befehlen (s. Kap. 4.1).

BEDIENUNGSHINWEISE

Das Programm wird mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das M-PGM generiert werden soll. Bei Eingabe von E wird es ans Basic-RAM-Ende gespeichert.

Der Basic-Loader löscht sich von selbst und die Uhr-Anzeige wird aktiviert. Außerdem wird angegeben, mit welchem SYS-Befehl nach Deaktivierung die Uhr "zurückgeholt" wird.

6.6 Uhr in der 24. Zeile

Wenn Sie z.B. eine 8 KByte-Speichererweiterung haben und dieses Programm als erstes ans RAM-Ende legen (per Eingabe von "E") so erhalten Sie die Aktivierungs-"Formel" SYS 16206. Nach Drücken der Tasten RUN/STOP und RESTORE wird also per SYS 16206 die Uhr-Anzeige wieder eingeschaltet.

Die VC-20-Uhr wird bekanntlich durch Eingabe von (Beispiel):

TI$="152456" für die Uhrzeit 15.24:56 Uhr

eingestellt.

```
10 REM VC-UHR
20 REM UHRZEIT IN DER 24.ZEILE
30 L=171
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>20996THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 SYSSD:NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS";SDCHR$(146):PRINT
280 PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
1000 DATA169,48,141,3,144,173,15,144,41,127,141,15,144,74,74,74
1010 DATA74,170,173,136,2,41,2,73,2,9,148,133,252,169,0,133
1020 DATA251,160,15,138,145,251,136,16,251,165,252,73,2,133,252,230
1030 DATA252,138,73,7,160,250,132,251,160,5,145,251,136,16,251,120
1040 DATA169,L75,141,20,3,169,H,141,21,3,96,165,162,41,192,205
1050 DATA251,3,240,84,141,251,3,162,114,181,0,157,64,3,202,16
1060 DATA248,162,16,189,0,1,157,224,3,202,16,247,165,162,166,161
1070 DATA164,160,32,135,207,32,75,207,169,250,133,0,174,136,2,232
1080 DATA134,1,32,166,214,170,160,0,232,202,240,7,177,34,145,0
1090 DATA200,208,246,162,16,189,224,3,157,0,1,202,16,247,162,114
1100 DATA189,64,3,149,0,202,16,248,76,191,234
```

Bild 6.6.1

```
2000 LDA #$30          24. Zeile oeffnen
2002 STA $9003
2005 LDA $900F
2008 AND #$7F
200A STA $900F         Mischhintergrundfarbe loeschen
200D LSR               Hintergrundfarbe in Bit 0-2 schieben
200E LSR
200F LSR
2010 LSR
2011 TAX               nach X sichern
2012 LDA $0288         Vektor  $FB,$FC  zum Farbspeicher fuer
2015 AND #$02              7.-22. Zeichen der 24. Zeile her-
2017 EOR #$02              stellen
2019 ORA #$94
201B STA $FC
201D LDA #$00
201F STA $FB
2021 LDY #$0F          In Zeile 24 fuer 7.-22. Zeichen:
2023 TXA                   Hintergrund- = Vordergrundfarbe
2024 STA ($FB),Y           setzen
2026 DEY
2027 BPL $2024
2029 LDA $FC           Vektor  $FB,$FC  auf Farbspeicher richten
202B EOR #$02
202D STA $FC
202F INC $FC
2031 TXA               Kontrast schaffen zwischen Uhrzeit-
2032 EOR #$07              anzeige und Hintergrundfarbe
2034 LDY #$FA          Vordergrundfarbe fuer die Uhrzeitanzeige
2036 STY $FB               herstellen
2038 LDY #$05
203A STA ($FB),Y
203C DEY
203D BPL $203A
203F SEI
2040 LDA #$4B          IRQ-Vektor verbiegen
2042 STA $0314
2045 LDA #$20
2047 STA $0315
204A RTS
204B LDA $A2           Uhrzeit-Ausgabe wird nur alle 64/60
204D AND #$C0              Sekunden durchlaufen
204F CMP $03FB
2052 BEQ $204A
2054 STA $03FB
2057 LDX #$72          Zero-Page-Adressen $00-$72 nach
2059 LDA $00,X             Kassettenpuffer sichern
205B STA $0340,X
205E DEX
205F BPL $2059
2061 LDX #$10          Stack-Adressen $0100-$0110 nach
2063 LDA $0100,X           Kassettenpuffer sichern
2066 STA $03E0,X
2069 DEX
206A BPL $2063
```

Bild 6.6.2 (Teil 1)

```
206C LDA $A2            Zeit holen
206E LDX #A1
2070 LDY #A0
2072 JSR $CF97          Zeit in ASCII-String umwandeln
2075 JSR $CF4B
2078 LDA ##rA           Vektor $00,$01 auf 1. Stelle der
207A STA $00                Zeile 24 richten
207C LDX $0288
207F INX
2080 STX $01
2082 JSR $D6A5          String-Deskriptor nach
2085 TAX                    A,$22,$23 (Anzahl,L,H)
2086 LDY #$00
2088 INX
2089 DEX
208A BEQ $2093
208C LDA ($22),Y        Uhrzeit ausgeben
208E STA ($00),Y
2090 INY
2091 BNE $2089
2093 LDX ##10           $0100-$0110 wieder zurueckspeichern
2095 LDA $03E0,X
2098 STA $0100,X
209B DEX
209C BPL $2095
209E LDX ##72           $00-$72 wieder zurueckspeichern
20A0 LDA $0340,X
20A3 STA $00,X
20A5 DEX
20A6 BPL $20A0
20A8 JMP $EABF          Ende der IRQ-Umleitung
```

Bild 6.6.2 (Teil 2)

PROGRAMM-EINZELHEITEN

1.) Name: VC-UHR

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Basic-Loader

4.) Anzahl der Bytes
des Basic-Loaders: 1575
des M-PGMs: 171

5a) Benutzte Variablen des Basic-Loaders: s. Kap. 5.2

5b) Benutzte Speicherzellen des M-PGMs:

$FB,FC	Hilfsvektor
$03FB	Zeitanteilzwischenspeicher
$0340-0340+$72	Zwischenspeicher der Zellen $00-72
$03E0-03F0	Zwischenspeicher der Zellen $0100-0110

6.) Listings

des Basic-Loaders:	Bild 6.6.1
des M-PGMs:	Bild 6.6.2 (Teil 1 u. Teil 2)

7.) Erlauterungen zum

Basic-Loader:	s. Kap 5.2
M-PGM:	implizit gegeben in Bild 6.6.2

6.7 SCHNELLE SUCHROUTINE FÜR STRINGFELDER

Leider muß es gerade eine so nutzliche Tatigkeit wie das Suchen sein, bei der sich zeigt, daß die Bearbeitungszeiten fast unertraglich lang werden, wenn die Suchroutine in Basic geschrieben ist. Hier muß also ein Maschinenprogramm her, welches man universell in jedem Basic-Programm unterbringen kann. Es heißt VC-SEARCH, ist nur 186 Bytes lang und durchsucht jedes beliebige Stringfeld bis zu 100mal schneller als ein vergleichbares Suchprogramm in Basic.

Das Maschinenprogramm VC-SEARCH (s. Bild 6.7.4) wird je nach Wunsch in einem beliebigen Speicherbereich generiert. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen Überschreiben geschutzt und sie konnen es auch dann aufrufen, wenn Sie beliebige andere Programme im Arbeitsspeicher haben. Weiterhin ist es koppelbar mit von Ihnen frei definierbaren Befehlen (s. Kap. 4.1).

6.7 Suchroutine für Stringfelder

```
10 REM VC-SEARCH
20 REM SUCHE IN EINEM FELD NACH EINEM STRING
30 L=186
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260
170 FORI=SDTOSD+L-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>22640THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"NACH INITIALIS. MIT:"
270 PRINTCHR$(18)"SYS"SD+173CHR$(146)
280 PRINT:PRINT"M-PGM-AUFRUF MIT:"
290 PRINTCHR$(18)"SYS0,(LV%),(SB$),(SF$)"CHR$(146):PRINT:PRINT"ODER"
300 PRINT:PRINT:PRINT"M-PGM-AUFRUF MIT:"
310 PRINTCHR$(18)"SYS"SDCHR$(157)",(LV%),(SB$),(SF$)"CHR$(146)
320 PRINT:PRINT"LV%=LAUFVAR. (INTEGER)SB$=SUCHBEGRIFF"
330 PRINT"SF$=STRINGFELDNAME":PRINT:PRINT
340 PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
1000 DATA32,115,0,32,158,205,36,13,48,14,165,71,133,163,165,72
1010 DATA133,164,32,81,226,76,L0,H,160,L81,169,H,32,L102,H,160
1020 DATA0,177,163,133,167,200,177,163,133,166,160,1,132,11,136,132
1030 DATA12,132,14,136,132,13,160,L69,169,H,72,152,72,165,167,72
1040 DATA165,166,72,76,24,210,32,L109,H,230,166,208,221,230,167,208
1050 DATA217,104,104,32,L98,H,160,0,165,167,145,163,200,165,166,145
1060 DATA163,96,160,58,169,196,140,0,3,141,1,3,96,160,2,177
1070 DATA71,133,253,136,177,71,133,252,136,177,71,133,251,162,0,202
1080 DATA160,255,200,196,183,240,24,232,228,251,240,32,177,187,32,L164
1090 DATAH,209,252,8,32,L164,H,40,240,232,152,240,234,208,224,152
1100 DATA240,10,208,173,72,138,72,152,170,104,168,104,96,169,76,133
1110 DATA0,169,L0,133,1,169,H,133,2,96
```

Bild 6.7.1

BEDIENUNGSHINWEISE

Das Maschinen-Programm VC-SEARCH ist in dem Basic-Loader eingebunden, der im Kapitel 5.2 beschriebenen wird. Detailinformation bezgl. diesem kann dort in Erfahrung gebracht werden. Der Basic-Loader wird normal mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das M-PGM generiert werden soll. Bei Eingabe von "E" wird es ans Basic-RAM-Ende gespeichert, wobei eventuell schon andere vorhandene M-PGMe in keinster Weise angetastet werden.

Um mit der Benutzung von VC-SEARCH besser vertraut zu werden, gehen wir wieder von einem Beispiel aus. Wie nehmen an, wir hätten einen VC-20 mit einer angeschlossenen 16KByte-Speichererweiterung. Wenn wir uns per Eingabe von "E" fürs M-PGM-Generieren am Basic-RAM-Ende entscheiden und sonst kein anderes M-PGM dort vorhanden ist, erscheint folgende Anzeige auf dem Bildschirm:

```
NACH INITIALISIERUNG MIT:   SYS 24556
M-PGM-AUFRUF MIT:           SYS0,(LV%),(SB$),(SF$)

ODER:

M-PGM-AUFRUF MIT:           SYS 24383,(LV%),(SB$),(SF$)

LV% = LAUFV. (INTEGER), SB$ = SUCHBEGRIFF, SF$ = STRINGFELDNAME
```

Bei der ersten Aufrufmöglichkeit muß irgendwann vor der 1. Suchaktion die Initialisierung SYS 24556 vorgenommen werden. Dies kann entweder vor dem Start eines Programms oder auch im Programm geschehen. Die Initialisierung erfüllt nur den Zweck, daß ein Sprungbefehl in Maschinensprache in den ersten drei Speicherzellen #0,1,2 installiert wird, welcher bei Aufruf SYS0,... zur eigentlichen Suchroutine VC-SEARCH führt. Der Vorteil dieser Aufrufmöglichkeit ist, daß die eigenen Programme grundsätzlich portabel gehalten werden können, weil es nur einen Suchbefehl gibt, nämlich SYS0,...

Die Portabilität kann allerdings dadurch eleganter erzielt werden, daß der SYS-Sprung der zweiten Aufrufmöglichkeit per VC-COMMAND (s. Kap. 4.1) mit einem selbst gewählten Befehlswort, z.B. SEARCH, gekoppelt wird. Man wird somit frei von jeglicher Adressenangabe. Die Suchroutine läßt sich auch direkt so aufrufen, wie es die Anzeige vorgibt. Bei beiden Aufrufmöglichkeiten ist darauf zu achten, daß hinter dem Sprungbefehl drei durch Kommata getrennte Variablennamen angegeben werden müssen, deren Namen zwar beliebig wählbar sind, nicht aber deren Typ.

Die erste Variable (LV%) muß eine vom Typ Integer sein. Sie enthält die Feldelement-Nummer, deren Wert in dreierlei Hinsicht für den Benutzer von Bedeutung ist:

- Vor dem Such-Aufruf wird der Variablen (LV%) derjenige Wert zugewiesen, von welchem String-Feldelement mit der entsprechenden Nummer ab die Suche gestartet werden soll. In den meisten Fällen wird das der Wert 0 sein.

- Nach Rückkehr aus der Suchroutine enthält (LV%) die Nummer desjenigen Feldelements, welches den gesuchten Suchbegriff enthält.

- Wenn nach Rückkehr aus der Suchroutine der Wert von (LV%) um 1 großer ist als die Dimension des zu durchsuchenden Stringfeldes, dann bedeutet dies: "Nicht gefunden".

Die zweite Variable (SB$) muß eine vom Typ String sein. Diese enthält den Suchbegriff. Hierbei ist es allerdings auch erlaubt, auf eine Variable zu verzichten und den Suchbegriff in Anführungsstrichen direkt anzugeben.

Der dritte Name (SF$) bezieht sich auf das zu durchsuchende Stringfeld. Nur die Angabe des Namens ist erlaubt und nicht die sonst gewohnte in Klammern stehende Elementnummernangabe.

BEISPIEL 1

Im Demo-Programm gemaß Bild 6.7.2 wird auf obiges VC-SEARCH-Generierungsbeispiel aufgebaut. Nachdem gemaß dem ersteren Aufrufverfahren VC-SEARCH initialisiert worden ist (Zeile 10), wird ein Stringfeld namens DA$ (Zeile 20) mit 501 Feldelementen (0 ist miteinbezogen) dimensioniert. Das vorletzte Element (damit die Schnelligkeit der Routine erkennbar wird) erhält den Inhalt "ZU SUCHENDER SUCHBEGRIFF GEFUNDEN" und der Suchbegriff SB$ soll "SUCHBEGRIFF" heißen (Zeile 30). Allen anderen Feldelementen wird ein zum Teil zufalliger String der Gesamtlänge von 16 Bytes zugewiesen. Dabei kommt der VC-20 zwischendurch mal ins Stocken, weil die Garbage-Collection-Routine Platz für die noch verbleibenden Daten schaffen muß. Die Suchaktion wird in Zeile 100 gestartet. Nach 1.3 Sekunden liefern die restlichen Zeilen dieses Programms die folgende Anzeige auf dem Bildschirm:

```
GEF. IN 1.3 SEK
IM FELDELEMENT: 499
AN DER  14. STELLE
```

Mit Recht kann hier wohl behauptet werden: "Gute Leistung !". Die in der Literatur desöfteren genannte INSTRING-Funktion kann mit VC-SEARCH ebenso realisiert werden. Der in Zeile 130 aufgeführte Term: PEEK(781)PEEK(183)+2 gibt die Position innerhalb des String-Feldelements an, ab der der Suchbegriff vorkommt.

```
0 REM DEMO-PGM FUER VC-SEARCH (16K-V)
10 N=500:SYS24556
20 DIMDA$(N)
30 DA$(N-1)="ZU SUCHENDER SUCHBEGRIFF GEFUNDEN":SB$="SUCHBEGRIFF"
40 FORI=1TON-2
50 H$=CHR$(RND(1)*26+65)
60 FORJ=1TO3:H$=H$+H$:NEXTJ:H$=H$+"SUCHWORT":PRINTI;H$
70 DA$(I)=H$:NEXTI:DA$(N)=DA$(1)
80 PRINTI;DA$(N-1):PRINTI+1;DA$(N)
90 TA=TI
100 I%=0:SYS0,I%,SB$,DA$:IFI%>NTHENPRINT"NICHT GEFUNDEN":END
110 PRINT:PRINT"GEF. IN "CHR$(18)(TI-TA)/60CHR$(146)"SEK"
120 PRINT"IM FELDELEMENT "CHR$(18)I%
130 PRINT"AN DER "CHR$(18)PEEK(781)-PEEK(183)+2CHR$(146)CHR$(157)". STELLE"
```

Bild 6.7.2

BEISPIEL 2

Das in Bild 6.8.3 gezeigte Demo-Vergleichsprogramm soll Ihnen zeigen, daß die zu Beispiel 1 analoge jedoch in Basic geschriebene Suchroutine nach wesentlich längerer Zeit ihr Ziel erreicht, natürlich mit gleichem Resultat. Sie werden erkennen, daß das Programm prinzipiell genauso wie das erste aufgebaut ist. Unterschiedlich ist folgendes:

Die Suchaktion wird in den Zeilen 130, 140 vorgenommen. Die String-Durchsuchung wird dabei vom Unterprogramm in den Zeilen 20 und 30 erledigt (Unterprogramme sollten nach Möglichkeit immer am Anfang eines Programms stehen, weil somit ein schnellerer Zugriff auf sie möglich ist). Wird dieses mit der Positionsnummer P ungleich 0 verlassen (die INSTRING-Funktion wird mit dem Wert in P auch gleich miterfüllt), so ist das String-Feldelement gefunden. Dieses relativ einfache Basic-Such-Unterprogramm ist für so manche Fälle sicherlich ausreichend, liefert allerdings bei großen Datenmengen ein zeitlich ungünstiges Ergebnis.

6.7 Suchroutine für Stringfelder

```
0 REM DEMO-VERGLEICHs-PGM FUER VC-SEARCH (16K-V)
10 GOTO40
20 P=0:FORK=1TOLEN(DH$(1))-LS+1:IFMID$(DA$(I),K,LS)=SB$THENP=K:RETURN
30 NEXTK:RETURN
40 N=500
50 DIMDA$(N)
60 DA$(N-1)="ZU SUCHENDER SUCHBEGRIFF GEFUNDEN":SB$="SUCHBEGRIFF"
70 FORI=1TON-2
80 H$=CHR$(RND(1)*26+65)
90 FORJ=1TO3:H$=H$+H$:NEXTJ:H$=H$+"SUCHWORT":PRINTI;H$
100 DA$(I)=H$:NEXTI:DA$(N)=DA$(1)
110 PRINTI;DA$(N-1):PRINTI+1;DA$(N)
120 TA=TI:LS=LEN(SB$)
130 FORI=0TON:GOSUB20:IFPTHEN160
140 NEXTI
150 IFI>NTHENPRINT"NICHT GEFUNDEN":END
160 PRINT:PRINT"GEF. IN "CHR$(18)(TI-TA)/60CHR$(146)"SEK"
170 PRINT"IM FELDELEMENT: "CHR$(18)I
180 PRINT"AN DER "CHR$(18)PCHR$(146)CHR$(157)". STELLE"
```

```
GEF. IN 34.1 SEK
IM FELDELEMENT:  499
AN DER  14 . STELLE
```

Bild 6.7.3

PROGRAMM-EINZELHEITEN

1.) Name: VC-SEARCH

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Stringfeld-Suchprogramm

4.) Anzahl der Bytes
des Basic-Loaders: 1898
des M-PGMs: 186
des Beispiel-PGM 1: 466
des " " 2: 545

5a) Benutzte Variablen des Basic-Loaders: s. Kap. 5.2

5b) Benutzte Adressen des M-PGMs:

$A3,A4	Vektor zur Laufvariablenadresse
$A6,A7	Inhalt der Laufvariablen
$FB,FC,FD	Deskriptor des momentanen Stringfeld-Elements

```
2000 JSR $0073      Naechstes Zeichen holen (nach Komma)
2003 JSR $D09E      Ausdruck auswerten
2006 BIT $0D
2008 BMI $2016      Sprung, wenn Ausdruck = String
200A LDA $47        Laufvariablen-Adresse
200C STA $A3            nach
200E LDA $48            $A3,$A4
2010 STA $A4
2012 JSR $E251      Nachfolgenden String holen und Deskrip-
2015 JMP $2000          tor nach $B7,$B8,$B9 (Laenge,L,H)
2018 LDY #$51       Fehleranzeige-Vektor verbiegen
201A LDA #$20
201C JSR $2066
201F LDY #$00       Laufvariableninhalt holen und
2021 LDA ($A3),Y        nach $A6,$A7
2023 STA $A7
2025 INY
2026 LDA ($A3),Y
2028 STA $A6
202A LDY #$01       Diverse Initialisierungen
202C STY $0A
202E DEY
202F STY $0C
2031 STY $0E
2033 DEY
2034 STY $0D
2036 LDY #$45       Ruecksprungadresse-1 nach Stack
2038 LDA #$20
203A PHA
203B TYA
203C PHA
203D LDA $A7        Laufvariableninhalt nach Stack
203F PHA
2040 LDA $A6
2042 PHA
2043 JMP $D21B      Adresse des naechsten Feldstrings suchen
2046 JSR $206D      Feldstring nach Suchstring durchsuchen
2049 INC $A6        Laufvariable um 1 erhoehen
204B BNE $202A
204D INC $A7
204F BNE $202A      Sprung immer
2051 PLA            Ruecksprungadresse loeschen
2052 PLA
2053 JSR $2062      Fehleranzeige-Vektor auf Default setzen
2056 LDY #$00       Momentanen Laufvariableninhalt der
2058 LDA $A7            Laufvariablen zuordnen
205A STA ($A3),Y
205C INY
205D LDA $A6
205F STA ($A3),Y
2061 RTS
2062 LDY #$3A       Fehleranzeige-Vektor auf Default setzen
2064 LDA #$C4
2066 STY $0300      Fehleranzeige-Vektor verbiegen
2069 STA $0301
206C RTS
```

Bild 6.7.4 (Teil 1)

```
206D LDY #$02
206F LDA ($47),Y     Feldstringadresse HIGH
2071 STA $FD         nach $FD
2073 DEY
2074 LDA ($47),Y     Feldstringadresse LOW
2076 STA $FC         nach $FC
2078 DEY             Y=0
2079 LDA ($47),Y     Anzahl Feldstringzeichen
207B STA $FB             nach $FB
207D LDX #$00        Feldstringzeiger initialisieren
207F DEX
2080 LDY #$FF        Suchstringzeiger initialisieren
2082 INY
2083 CPY $B7
2085 BEQ $209F       Sprung, w. Suchstringende ueberschritten
2087 INX
2088 CPX $FB
208A BEQ $20AC       RTS, wenn Feldstringende ueberschritten
208C LDA ($BB),Y
208E JSR $20A4       X,Y vertauschen
2091 CMP ($FC),Y
2093 PHP             Z-Flag retten
2094 JSR $20A4       X,Y vertauschen
2097 PLP             Z-Flag zurueckholen
2098 BEQ $2082       Sprung, wenn Zeichen identisch
209A TYA
209B BEQ $2087       Sprung, wenn Y=0
209D BNE $207F       Sprung, wenn Y<>0
209F TYA
20A0 BEQ $20AC       RTS, wenn Laenge des gefund. strings = 0
20A2 BNE $2051       Suchroutine verlassen
20A4 PHA
20A5 TXA
20A6 PHA
20A7 TYA             X,Y vertauschen
20A8 TAX
20A9 PLA
20AA TAY
20AB PLA
20AC RTS
20AD LDA #$4C        "JMP $(Anfangsadresse)"
20AF STA $00             nach $00,$01,$02
20B1 LDA #$00
20B3 STA $01
20B5 LDA #$20
20B7 STA $02
20B9 RTS
```

Bild 6.7.4 (Teil 2)

5c) Benutzte Variablen in den Beispiel-Programmen:

N	Maximale Feldelement-Nummer
SB$	Suchbegriff
DA$()	Stringfeld
H$	Zufälliger Buchstabe
I%	Laufvariable zur Indizierung der DA$-Feldelemente
LS	Länge von SB$
I,J	Schleifenzähler
TA	VC-20-Zeit bei Suchbeginn

6.) Listings

des Basic-Loaders:	Bild 6.7.1
des M-PGMs:	Bild 6.7.4 (Teile 1 und 2)
der Beispiel-PGMe:	Bilder 6.7.2 und 6.7.3

7.) Erläuterungen zum Basic-Loader werden in Kap. 5.2 gegeben und zum M-PGM implizit im Listing (Bild 6.7.4).
Die Beispiel-Programme wurden bereits oben näher erläutert.

6.8 SOFTWARESCHUTZ

Gerichtsurteile der letzten Zeit haben gezeigt: Software ist geistiges Eigentum des Programmierers und ist durch das Urheberrecht geschützt. Trotzdem werden Programme kopiert und ohne Erlaubnis des Verfassers weitergegeben (verkauft). Die Folge ist, daß fast alle PGMe mit einer kleinen Routine versehen sind, die ein Auflisten, Andern oder Kopieren verhindern sollen.

Hier soll anhand von Beispielen gezeigt werden, wie Sie Ihre Programme gegen unerlaubtes Auflisten oder Kopieren schützen können.

LISTSCHUTZ

Unter Listschutz versteht man den Schutz von Software gegen unerlaubtes Auflisten. Es können komplette PGMe oder PGM-Teile geschützt werden.

6.8 Softwarschutz

Variante 1:

Vielleicht haben Sie auch schon einmal versucht, ein PGM zu LISTen und bekamen nach der LIST-Anweisung die Fehlermeldung: "SYNTAX ERROR". Das lag daran, daß versucht wurde, ein Zeichen zu listen, das nicht als Basic-Code definiert war und auch nicht in Anführungszeichen stand.

Geben Sie als Versuch folgende Zeilen ein:

```
10 REM:(Shift)L
20 PRINT "OK"
30 END
```

Shift L in Zeile 10 bedeutet: L mit gedrückter SHIFT-Taste eingeben. Wenn Sie dieses kleine PGM starten, werden Sie feststellen, daß es fehlerfrei arbeitet. Ein Versuch, das PGM aufzulisten, wird jedoch die Fehlermeldung "SYNTAX ERROR" erzeugen. Diesen LISTschutz können Sie, so oft Sie wollen und an jeder Stelle Ihres PGMs einbauen.

Variante 2:

Der LISTschutz nach Variante 1 ist schnell zu entdecken und leicht zu entfernen. LISTschutz-Variante 2 ist vielleicht genauso schnell zu durchschauen, aber gar nicht so einfach zu entfernen.

Geben Sie dazu in alle Zeilen, die geschützt werden sollen, direkt hinter der Zeilennummer 5 Doppelpunkte ein.

Beispiel:

```
100 :::::PRINT "LISTSCHUTZ"
110 END
```

Die folgenden drei Basic-Zeilen müssen an das zu schützende PGM angehängt und mit GOTO 60000 gestartet werden.

```
60000 FOR J=PEEK(43)+PEEK(44)*256 TO PEEK(45)+
      PEEK(46)*256
60001 IF PEEK(J)=58 AND PEEK(J+1)=58 AND PEEK(
      J+2)=58 AND PEEK(J+3)=58 AND PEEK(J+4)=
      58 THEN POKE J,0:J=J+4
60002 NEXT
```

Das PGM ab Zeile 60000 sucht nach den 5 Doppelpunkten und schreibt (POKE) an die Adresse des ersten Doppelpunktes eine Null. Nach READY ist das PGM geschützt und kann, nachdem die Zeilen 60000 bis 60002 gelöscht wurden, geSAVEd werden. Wenn jetzt das PGM geLISTet wird, dann wird zwar die Zeilennummer, aber nicht der Inhalt dieser Zeile aufgeLISTet.

Was ist passiert?
Der erste Doppelpunkt hinter der Zeilennummer wurde durch die POKE-Anweisung in 0 geändert. Diese 0 wird von der LIST-Routine als Zeilenendekennzeichen erkannt und die nächste Zeile, deren Anfangsadresse bekannt ist, wird aufgelistet (vgl. VC-20-Programmier-Handbuch S. 122: "Speicherung von Basic-Anweisungen" sowie Kap. 2.1 "Speicherorganisation bei einem Basic-Programm"). Im PGM-Ablauf wird die 0 hinter der Zeilennummer auch als Zeilenendekennzeichen erkannt, aber die folgenden vier Doppelpunkte werden als Koppeladresse und Zeilennummer interpretiert. Die dadurch folgenden Befehle und Anweisungen werden fehlerfrei abgearbeitet.

Variante 3:

Der LISTschutz nach Variante 3 ist beim Auflisten des PGMs gar nicht als solcher zu erkennen und deshalb sehr wirkungsvoll. Zum besseren Verständnis soll auch dieser LISTschutz wieder anhand eines kleinen Beispiels beschrieben werden.
Geben Sie dazu folgende Zeilen ein:

```
10 A$="TEST":GOTO 1000:REM""
20 A$=""
1000 PRINT A$
1010 END
```

Danach führen Sie den Cursor auf das zweite Anführungszeichen hinter REM in Zeile 10 und drücken bei gedrückter SHIFT-Taste 15mal die Taste INST/DEL. Dadurch schaffen Sie Platz für 15 Zeichen, die eingefügt werden müssen. Drücken Sie jetzt 15mal die Taste INST/DEL ohne SHIFT. Der eben geschaffene Platz wird mit dem Steuerzeichen für DELete (invertiertes T) gefüllt. Anschließend wird die RETURN-Taste betätigt. Die zweite Hälfte der Zeile 10 ist jetzt mit einem LISTschutz versehen. Die LIST-Anweisung wird die Zeile 10 komplett auf dem Bildschirm darstellen, aber

die 15 Steuerzeichen für DEL löschen sofort die 15 letzten Zeichen dieser Zeile. Das geht so schnell, daß unser träges Auge es gar nicht wahrnimmt.

KOPIERSCHUTZ

Eine weitere Möglichkeit, Software zu schützen, besteht darin, das PGM mit einem Kopierschutz zu versehen. D.h.: nach dem unerlaubten Kopieren soll ein auf diese Weise geschütztes PGM gar nicht mehr oder fehlerhaft arbeiten.

Einen Kopierschutz kann man realisieren, indem das PGM unter einem Namen auf Kassette gespeichert wird, der länger als 16 Zeichen ist und im PGM-Ablauf den Inhalt des Kassettenpuffers mit dem PGM-Namen vergleicht. Wie Sie wissen (Kap. 3.2), ist es möglich, ein PGM unter einem Namen zu SAVEn, der max. 176 Zeichen lang ist. Von diesem PGM-Namen werden bei LOAD nur 16 Zeichen dargestellt und es bietet sich geradezu an, die verbleibenden 160 Bytes zu Zwecken des Softwareschutzes auszunutzen.

Das folgende kleine Beispiel-PGM muß unter dem 17 Zeichen langen Namen "KOPIERSCHUTZDEMOA" auf Kassette geSAVEd werden.

```
10 IF PEEK(849) <> 65 THEN STOP
20 PRINT"OK"
30 END
```

Zeile 10 des PGMs prüft das 17. Zeichen des PGM-Namens im Kassettenpuffer und stoppt den PGM-Ablauf, wenn ein Zeichen ungleich "A" erkannt wird.

Sie können auch komplette Maschinensprache-Programme im PGM-Namen verbergen, die bei LOAD mit in den Kassettenpuffer geladen werden. Schreiben Sie dazu das M-PGM zusammen mit dem PGM-Namen in den Kassettenpuffer. Die folgende Zeile, im Direkt-Modus eingegeben, schreibt den Inhalt des Kassettenpuffers in die Variable N$ und SAVEd das PGM unter dem Namen N$ auf Kassette.

```
N$="":FOR J=833 TO 1019:N$=N$+CHR$(PEEK(J)):NEXT:SAVE N$
```

WEITERE SCHUTZMÖGLICHKEITEN

Alle bisher beschriebenen Software-Schutzmaßnahmen, die naturlich auch kombiniert werden können, sind kein großes Hindernis für jemanden, der weder Zeit noch Mühen scheut, den Softwareschutz zu durchbrechen. Deshalb werde an dieser Stelle der Hinweis auf das Kap. 6.5 "Autostart für Basic- und Modulprogramme" gegeben. Dem Anwender werden dort Moglichkeiten aufgezeigt, die es ihm gestatten, den fast 100%igen Softwareschutz zu realisieren.
Man kann z.B.:

- wahrend des Ladevorgangs die Tastatur vollständig verriegeln, so daß die STOP- und auch die RESTORE-Taste keine Funktion mehr haben (s. Kap. 2.7 "Tastaturabfrage und -verriegelung") oder

- wahrend des Ladevorgangs den Vektor "Basic-Beginn" andern und das PGM mitten in den verfügbaren RAM-Bereich laden. Nach einem Hardware-RESET kann selbst der Befehl OLD (s. Kap. 4.5) das PGM nicht mehr retten, weil die Startadresse des PGMs nicht bekannt ist.

Versuchen Sie es doch mal, ein derart supergeschütztes Programm selbst zu schreiben.

6.9 AUTOMATISCHE STRING-EINGABE: AUTOINPUT

Stellen Sie sich vor, Sie wurden bei der Programmerstellung immer wieder auf die gleichen Unterprogramme, ganze Programmteile oder Basic-Makros zugreifen wollen, dies aber nach Moglichkeit ohne große Mühen. Mit dem Programm VC-AUTOINPUT können Sie eine solche Idee relativ einfach verwirklichen. Die Eingabe einer SYS-Adresse oder eines selbst definierten Wortes reicht aus, um Programmteil- oder Direkt-Modus-Eingaben automatisch eingeben zu lassen.

VC-AUTOINPUT ist ein Maschinenprogramm (s. Bild 6.9.6), welches mehrere von Ihnen vorgegebene Eingaben beherbergt und je nach Wunsch in einem beliebigen Speicherbereich generiert wird. Wenn Sie es pauschal am RAM-Ende ablegen lassen, wird es gegen Überschreiben geschutzt und sie können es auch dann aufrufen, wenn

Sie beliebige andere Programme im Arbeitsspeicher haben. Weiterhin ist jede einzelne Eingabe mit von Ihnen frei festlegbaren Aufrufbegriffen (s. Kap. 4.1) koppelbar.

BEDIENUNGSHINWEISE

Das Maschinenprogramm VC-AUTOINPUT ist in einem Basic-Loader (s. Bild 6.9.1) eingebunden, der im Kapitel 5.2 beschrieben ist. Detailinformation über ihn kann dort in Erfahrung gebracht werden. Allerdings kommen hier speziell beim VC-AUTOINPUT-Basic-Loader noch einige besondere Dinge hinzu, die einer näheren Erläuterung bedürfen.

Irgendwie müssen ja die von Ihnen vorgegebenen Texte, Programmteile oder Makros - wir gebrauchen für diese Begriffe zwecks Vereinheitlichung im weiteren den Namen "Autoinputs" - mit dem M-PGM VC-AUTOINPUT kombiniert werden, so daß sie gemeinsam eine zusammenhängende Einheit werden und als solche nach Belieben überallhin abgespeichert werden können. Hierfür gibt es einige Regeln, die in Kurzform in den Zeilen 20 und 25 des Basic-Loaders (Bild 6.9.1) zu finden sind:

- Die Autoinputs müssen vor Zeile 2000 als Strings oder String-Teile in DATA-Anweisungen untergebracht werden.

- Falls mehrere Autoinputs im Basic-Loader untergebracht werden sollen, muß jeder mit dem "Klammeraffen"-Symbol als Schlußkennzeichen abgeschlossen werden. Der letzte Autoinput wird durch zwei weitere Klammeraffen beendet, so daß also hinter diesem insgesamt 3 Klammeraffen-Symbole stehen. Falls das Klammeraffen-Symbol selbst als ein Autoinput-Zeichen auftritt, so muß unmittelbar hinter diesem ein zweites folgen.

- Nicht immer ist bei Autoinputs ein abschließendes CR-Zeichen (RETURN-Taste) erwünscht, denken wir nur an Programmzeilenteile, die durch Zusätze individuell verändert werden. Wenn aber ein CR-Zeichen verlangt wird, so wird dies im Autoinput durch das Linkspfeil-Zeichen zum Ausdruck gebracht. Falls das Linkspfeil-Zeichen selbst Bestandteil des Autoinputs ist, so muß unmittelbar hinter diesem ein zweites folgen.

```
10 REM VC-AUTOINPUT
20 REM AUTOM. EINGABE VON TEXTEN IN DATA'S AB 1000
25 REM "←"=RETURN, "←←"="←"; "@"=ENDE EINES TEXTES, "@@ "="@", "@@@"=GESAMTENDE
30 L=88:GOSUB500:ML=L:L=L+3*BZ+ZZ
40 PRINTCHR$(147)"ABLEGEN DES M-PGM:"
50 PRINT:PRINT:PRINT"E = AM ENDE VOM BASIC-"SPC(4)"RAM-BEREICH"
60 PRINT:PRINT:PRINT"(ZAHL) GEWUENSCHTE AN-"SPC(7)"ANFANGSADRESSE"SPC(8)"(DEZIMA
L) VOM"
70 PRINTSPC(7)"PROGRAMM":PRINT:PRINT
80 INPUTA$:SD=VAL(A$):IFSDTHEN160
90 IFA$<>"E"THEN40
100 EM=PEEK(55)+256*PEEK(56):AD=EM-L:ER=PEEK(643)+256*PEEK(644)
110 IFEM=ERTHEN130
120 IFPEEK(ER-3)<>197ORPEEK(ER-2)<>206ORPEEK(ER-1)<>196THEN150
130 POKEEM-3,197:POKEEM-2,206:POKEEM-1,196:AD=AD-7:GOSUB250
140 POKEEM-5,AL:POKEEM-4,AH%
150 GOSUB250:POKE55,AL:POKE56,AH%:AL=FRE(9):SD=AD
160 GOSUB260:FORI=0TOBZ-1:POKESD+I*3,44:POKESD+I*3+1,169:POKESD+3*I+2,I:NEXT
165 SD=SD+3*BZ:PF=1:RESTORE:GOSUB500
170 FORI=SDTOSD+ML-1:READA$
180 IFA$<"H"THENA=VAL(A$):CH=CH+A:GOTO210
190 IFA$>"H"THENAD=VAL(RIGHT$(A$,LEN(A$)-1)):CH=CH+AD:AD=AD+SD:GOSUB250:A=AL:GOT
O210
200 A=AH%
210 POKEI,A:IFPEEK(I)=ATHENNEXT:GOTO230
220 PRINT:PRINT:PRINT"ROM-BEREICH !":PRINT"ANDERE WAHL":PRINT"FUER ANFANGSADRESS
E":STOP
230 IFCH<>9860THENPRINT:PRINT"PROGRAMM IST FEHLER-  HAFT EINGEGEBEN !!!":STOP
240 NEW
250 AH%=AD/256:AL=AD-256*AH%:RETURN
260 PRINTCHR$(147)"M-PGM-AUFRUF MIT:":PRINT
270 PRINTCHR$(18)"SYS"SD-2CHR$(157)"+3*N"CHR$(146):PRINT
280 PRINT"N=1 FUER 1. TEXT"
290 PRINT"N=2 FUER 2. TEXT"
300 PRINT"ETC.":PRINT:PRINT
310 PRINT"WARTEN AUF "CHR$(18)"READY"CHR$(146)"!!":RETURN
500 BZ=0:ZZ=0:FE=0:LD=0:I=0
510 GOSUB600:A1$=H$:GOSUB600:A2$=H$:GOSUB600
520 GOSUB700:IFPFTHENPOKESD+ML+ZZ,P
530 IFP=0THENBZ=BZ+1
540 ZZ=ZZ+1:IFFE=0THENGOSUB750:GOTO520
550 RETURN
600 IFI<>LDTHENI=I+1:H$=MID$(E$,I,1):RETURN
610 LD=0:I=0:READE$:LD=LEN(E$):IFLD=0THEN610
620 GOTO600
700 IFA1$<>"@"THEN800
710 IFA2$="@"ANDH$="@"ANDLD=ITHENP=0:FE=1:RETURN
720 IFA2$<>"@"THENP=0:RETURN
730 P=64:GOSUB750:RETURN
750 A1$=A2$:A2$=H$:GOSUB600:RETURN
800 IFA1$<>"←"THENP=ASC(A1$):RETURN
810 IFA2$="←"THENP=95:GOSUB750:RETURN
820 P=13:RETURN
1000 DATA @@@
2000 DATA162,L88,160,H,134,163,132,164,170,240,16,32,L83,H,240,5
2010 DATA32,L77,H,208,251,32,L77,H,202,208,240,162,L39,160,H,120
2020 DATA142,20,3,140,21,3,96,32,L83,H,240,10,32,L64,H,176
2030 DATA12,32,L77,H,208,246,162,191,160,234,32,L31,H,76,191,234
2040 DATA166,198,236,137,2,176,5,157,119,2,230,198,96,230,163,208
2050 DATA2,230,164,160,0,177,163,96
```

Bild 6.9.1

- Was die Benutzung von Anführungsstrichen sowie der Zeichen Doppelpunkt und Komma als Bestandteil eines Autoinputs angeht, sind die eigens dafür in Kapitel 2.6 aufgeführten Regeln zu beachten.

Ist der Basic-Loader dann entsprechend obiger Regeln in individueller Weise von Ihnen präpariert worden, wird dieser normal mit RUN gestartet. Sie werden gefragt, ab welcher Adresse (dezimal) das M-PGM zusammen mit den Autoinputs generiert werden soll. Bei Eingabe von "E" wird es ans Basic-RAM-Ende gespeichert, wobei eventuell schon andere vorhandene M-PGMe in keinster Weise angetastet werden.

BEISPIEL 1

Wir gehen beispielsweise davon aus, daß wir einen VC-20 in Grundversion (GV) haben. VC-AUTOINPUT soll uns jetzt die Arbeit der Eintipperei von mehreren PEEK-Befehlen zwecks Ausgabe der in Kapitel 2.1 erlauterten Basic-Arbeitsspeicher-Vektoren abnehmen. Dazu fügen wir die in Bild 6.9.2 aufgelisteten Zeilen 1000-1050 in den Basic-Loader aus Bild 6.9.1 ein. Beachten Sie bitte hierbei, daß für PRINT und PEEK die im VC-20-Handbuch auf S. 133 aufgeführten Anweisungs-Kurzel verwendet worden sind, um Speicherplatz zu sparen.

```
1000 DATA "?:",?"BA"SPC(2),"P"(43);P"(44)SPC(2)P"(43)+256*P"(44)←@"
1010 DATA "?:",?"VA"SPC(2),"P"(45);P"(46)SPC(2)P"(45)+256*P"(46)←@"
1020 DATA "?:",?"FA"SPC(2),"P"(47);P"(48)SPC(2)P"(47)+256*P"(48)←@"
1030 DATA "?:",?"FE"SPC(2),"P"(49);P"(50)SPC(2)P"(49)+256*P"(50)←@"
1040 DATA "?:",?"BE"SPC(2),"P"(55);P"(56)SPC(2)P"(55)+256*P"(56)←@"
1050 DATA @@
```

Bild 6.9.2

Wie Sie den Zeilen in Bild 6.9.2 leicht entnehmen können, handelt es sich hierbei um insgesamt 5 Autoinputs, denn das letzte Zeichen jeder der DATA-Zeilen 1000-1040 ist ein Klammeraffen-Zei-

chen. In diesem Beispiel ist dies reiner Zufall. Es ist ebenso möglich mehrere Autoinputs in eine einzige DATA-Zeile zu packen. Genauso ist es erlaubt, daß sich ein Autoinput über mehrere DATA-Zeilen erstreckt. Innerhalb einer DATA-Zeile können Sie, so oft Sie wollen, Stringteile durch Kommata trennen.

Dieser von uns präparierte VC-AUTOINPUT-Basic-Loader wird nun mit RUN gestartet (vorher SAVEn !!). Nach einer kurzen Berechnungszeit geben wir "E" ein, damit das M-PGM mit den Autoinputs (s. Bild 6.9.5) am Basic-RAM-Ende abgespeichert wird. Wenn dort sonst kein anderes M-PGM vorhanden ist, wird folgende Anzeige auf dem Bildschirm erscheinen:

M-PGM-AUFRUF MIT: SYS 7313 + 3*N

N=1 für 1. Text, N=2 für 2. Text, etc.

Wir wissen bereits, daß wir 5 Autoinputs abgespeichert haben. Demnach sind gemäß der angezeigten Erläuterung die Werte N=1, N=2, N=3, N=4 und N=5 möglich. Falls wir in der Basic-Loader-Zeile 240 das NEW durch ein END ersetzt hatten, würden wir nach Eingabe der 5 SYS-Adressen das in Bild 6.9.3 zu sehende Resultat erhalten. Es versteht sich von selbst, daß anstatt SYS 7313+3*1 ebenso SYS 7316 eingegeben werden kann. Auch können diese Adressen Inhalte eines Feldes AT(5) sein, so daß die Eingaben per SYS AT(1), SYS AT(2) etc. erlaubt sind. Diese SYS-Befehle dürfen keineswegs nur im Direkt-Modus benutzt werden. Auch innerhalb von Programmen sind sie gültig.

Eingabe:	Ausgabe auf dem Bildschirm:			
SYS 7313+3*1	BA	1	16	4097
SYS 7313+3*2	VA	210	24	6354
SYS 7313+3*3	FA	108	25	6508
SYS 7313+3*4	FE	108	25	6508
SYS 7313+3*5	BE	147	28	7315

Bild 6.9.3

Was als Resultat in diesem Beispiel herauskommt, sind die Werte und deren LOW/HIGH-Anteile der Vektoren: Basic-Anfang (BA), Variablen-Anfang (VA), Felder-Anfang (FA), Felder-Ende (FE) und Basic-Ende (BE). Der Vollständigkeit halber sollte erwähnt werden, daß über den in Bild 6.9.3 aufgezeigten Bildschirmausgaben noch zusätzlich der jeweilige Autoinput selbst zu sehen ist.

BEISPIEL 2

Nehmen wir an, wir gebrauchen in unserer Programmerstellung häufig die Eingabe-Warteschleife nach folgendem Muster:

```
10 GET E$:IF E$="" THEN 10
```

Es wäre also wünschenswert, wenn wir per VC-AUTOINPUT das "Gerippe" dieser Warteschleife auf den Bildschirm zaubern könnten, um nach M-PGM-Aufruf nur noch die passende Zeilennummer nachtragen zu müssen. Die Einbringung der Zeilen 1000 und 1010 gemäß Bild 6.9.4 in den Basic-Loader (Bild 6.9.1) führen uns zur Realisierung dieser Idee. Nach Generierung von VC-AUTOINPUT in gleicher Weise wie oben, läßt sich, wie vorhergesehen, das Warteschleifen-Makro per SYS 7562 hervorholen.

```
1000 DATA "   GETE$:IF",E$=""THEN@
1010 DATA @@
```

Bild 6.9.4

PROGRAMM-EINZELHEITEN

1.) Name: VC-AUTOINPUT

2.) Ausbaustufe: beliebig

3.) Art des PGMs: Betriebssystem-Zusatzroutine

4.) Anzahl der Bytes
des Basic-Loaders: 1938 (ohne Autoinputs)
des M-PGMs: 88 (" ")
des Beispiel-PGM 1: 2257
des " " 2: 1968

5.) Benutzte Variablen: s. Kap. 5.2
sowie die in den Zeilen 500-820 vorkommenden Hilfsvariablen

6.) Listings
des Basic-Loaders: Bild 6.9.1
des M-PGMs: Bild 6.9.6
der Autoinput-Tabelle
im Beispiel 1: Bild 6.9.5

7.) Die wesentlichen Erläuterungen zum Basic-Loader werden in Kap. 5.2 gegeben. Speziell beim VC-AUTOINPUT-Basic-Loader kommen die Zeilen 500-820 hinzu, die für die ordnungsgemäße Integrierung der Autoinputs in das M-PGM sorgen.
Die Erläuterungen zum M-PGM sind implizit im Listing (Bild 6.9.6) gegeben.

2067	3F 3A 3F 22 42	? : ? " B
206C	41 22 53 50 43	A " S P C
2071	28 32 29 50 C5	(2) P ─
2076	28 34 33 29 3B	(4 3) ;
207B	50 C5 28 34 34	P ─ (4 4
2080	29 53 50 43 28	) S P C (
2085	32 29 50 C5 28	2) P ─ (
208A	34 33 29 2B 32	4 3) + 2
208F	35 36 2A 50 C5	5 6 * P ─
2094	28 34 34 29 0D	(4 4) CR
2099	00	END

Bild 6.9.5: Die am M-PGM-Ende generierte Tabelle im Beispiel 1

```
2000 BIT $00A9          = LDA #$00  bei Sprung in $2001
2003 LDX #$5B           Scanner-Vektor  $A3,$A4  auf
2005 LDY #$20               Textanfang stellen ($205B)
2007 STX $A3
2009 STY $A4
200B TAX                Z-Flag beeinflussen
200C BEQ $201E          Sprung, wenn 1. Text ausgesucht
200E JSR $2056          Momentanes Zeichen holen
2011 BEQ $2018          Sprung, wenn Textende gefunden
2013 JSR $2050          Textende suchen
2016 BNE $2013          zurueck, wenn noch nicht gefunden
2018 JSR $2050          Scanner auf naechstes Zeichen stellen
201B DEX
201C BNE $200E          ausgesuchter Text noch nicht gefunden
201E LDX #$2A           Interruptroutine umleiten
2020 LDY #$20
2022 SEI
2023 STX $0314
2026 STY $0315
2029 RTS
202A JSR $2056          Gegenwaertiges Zeichen holen
202D BEQ $2039          Sprung, wenn Textende
202F JSR $2043          Zeichen nach Tastaturpuffer (Versuch)
2032 BCS $2040          Sprung, wenn Tastaturpuffer voll
2034 JSR $2050          Naechstes Zeichen holen
2037 BNE $202F          Sprung zurueck, wenn nicht Textende
2039 LDX #$BF           Interruptroutine auf alten Stand bringen
203B LDY #$EA
203D JSR $2022
2040 JMP $EABF
2043 LDX $C6
2045 CPX $0289          Wieviel Zeichen im Tastaturpuffer?
2048 BCS $204F          Sprung raus, wenn Puffer voll
204A STA $0277,X        Zeichen in Puffer ablegen
204D INC $C6
204F RTS
2050 INC $A3             $A3,$A4 = $A3,$A4 +1
2052 BNE $2056
2054 INC $A4
2056 LDY #$00
2058 LDA ($A3),Y        Zeichen holen
205A RTS
205B BRK                entspricht #$00 = Textende
```

Bild 6.9.6

6.10 BIT-MAPPING BEIM VC-20

Jeder Benutzer des VC-20 weiß es: das Betriebssystem des VC-20 unterstützt die hochauflosende Graphik leider nicht. Befehle, wie "Zeichne ein Kreis" oder "Zeichen eine Gerade" (alles naturlich in gewohnter Weise in Englisch), sind also von Hause aus nicht vorhanden. Dies ist nicht weiter tragisch, weil der VC-20 dem Be-

nutzer auch in dieser Hinsicht glücklicherweise viel "Spiel"-Raum läßt. Bit-Mapping ist also hier das Stichwort und es stellt sich die Frage, wie das beim VC-20 realisiert wird. Für diejenigen, die den Begriff noch nicht kennen: Bit-Mapping heißt, jedes auf dem Bildschirm sichtbare Pixel nach Belieben "ein-" und "ausschalten" zu konnen (im Normalfalle besteht ein VC-20-Zeichen aus 8*8=64 Pixels). Darüberhinaus stellen wir die Forderung, daß die hierzu erarbeiteten Hilfsmittel für jegliche Anwendung, bei der Bit-Mapping gebraucht wird, bequem nutzbar sind.

ALLGEMEINES

Pirschen wir uns an die Bit-Mapping-Lösung derart heran, daß wir zuerst einmal sämtliche benötigte Angaben über die Beziehungen zwischen den Begriffen: Zeichengenerator, Videospeicher, deren Inhalte und Zeichengroße in sinnvoller und praktisch verwertbarer Form zusammentragen. Ohne ein wenig Theorie und Mathematik geht's leider nicht. Nur fertige Programmteile Ihnen vor die Füße zu werfen, soll nicht Sinn und Zweck dieses Kapitels sein. Vielmehr geht es hier darum, einen tieferen Blick in die Arbeitsweise des Bit-Mapping zu vermitteln, so daß Sie eigenständig auch andere Programmlösungen kreieren können und sogar vielleicht das Wissen auf Maschinenprogramme anwenden. Denn eines muß gleich von vornherein gesagt werden: das Bit-Mapping per Basic geht nicht besonders schnell. Ein Maschinenprogramm könnte bis zur 100fachen Geschwindigkeit die gleiche Arbeit verrichten.

Wenden wir uns zu Anfang dem Zeichengenerator zu. Dieser Speicherbereich ist im Normalfall, also nach Einschalten des VC-20 oder nach RESET (das Vorhandensein von Modulprogrammen schließen wir hierbei aus), der Bereich $8000-8FFF = 32768-36863 und umfaßt die Pixelzeilen aller darstellbaren VC-20-Zeichen. Unter einer Pixelzeile wird eine der 8 Zeilen verstanden, die jeweils aus 8 einzelnen Pixels bestehen. Die Pixelzeilen ergeben, untereinander geschichtet, ein Zeichen. Wenn das D-Bit (Bit 0 der Adresse $9003) gesetzt ist, besteht ein Zeichen nicht aus 8, sondern aus 16 Pixelzeilen. Die Kombination der in einer Pixelzeile vorhandenen Pixels wird durch den Wert eines Bytes (8 Bits = 8 Pixels) vollständig beschrieben. Deswegen sind die Pixelzeilen als Bytes gespeichert, 8 Bytes pro Zeichen. Der VC-20 beinhaltet 2 Zeichensätze von je 128 verschiedenen Zeichen (s. S. 141-142 im VC-20-

Handbuch). Dies ergibt 8*2*128 = 2048 Bytes. Da alle Zeichen auch in invertierter Form vorliegen, werden also 4096 Bytes benotigt, die ja genau in den Bereich $8000-8FFF hineinpassen.

Beim Bit-Mapping können wir mit dem vorgegebenen Zeichengenerator allerdings nichts anfangen. Wir müssen also unseren eigenen schaffen, was jedoch nur im RAM-Bereich möglich ist. Die Tabelle A.5.8 im Anhang A.5 gibt uns darüber Auskunft, wie die Zeichengenerator-Anfangsadresse festgelegt wird:

Zeichengenerator-Anfangsadresse ZA:

Festlegen: `POKE 36869,(PEEK(36869) AND 240) OR X` (1)

mit:

X	ZA
12	4096
13	5120
14	6144
15	7168

Daraus ergibt sich die folgende Beziehung zwischen dem Inhalt der Zelle 36869 und ZA:

Bestimmen: `ZA = (PEEK(36869) AND 3)*1024 + 4096` (2)

Darüberhinaus ist die Festlegung der Anzahl der Pixelzeilen pro Zeichen notwendig. Es gibt nur die bereits oben genannten Moglichkeiten, nämlich 8 oder 16, die durch das D-Bit des VIC-Registers CR3 (s. Tabelle A.5.10 im Anhang A.5) bestimmt werden.

Pixelzeilen-Anzahl PZN:

Festlegen: `POKE 36867,(PEEK(36867) AND 254) OR D` (3)

mit:

D	PZN
0	8
1	16

Bestimmen: `PZN = 8*(1 + (PEEK(36867) AND 1))` (4)

Das, was auf dem Bildschirm zu sehen ist, stellt gewissermaßen ein bestimmter visualisierter Speicherbereich des VC-20 dar, fragt sich nur welcher. Wie oben stellen wir auch hierfur einige Beziehungen her:

Videospeicher-Anfangsadresse VA:

Festlegen: siehe Tabelle A.5.6 im Anhang A.5

Bestimmen:
```
VA = (PEEK(36869) AND  48)*64 +                    (5)
     (PEEK(36866) AND 128)* 4 + 4096
```

Zum Beispiel ist nach dem Einschalten VA = 7680 für einen VC-20 in Grundversion und VA = 4096 für einen mit einer angeschlossenen Speichererweiterung von mehr als 3 KByte.

In Verbindung mit dem Videospeicher sind noch zwei weitere Größen von Bedeutung:

CN = Anzahl der Zeichen pro Zeile:

```
Festlegen:    POKE 36866,(PEEK(36866) AND 128) OR CN     (6)
Bestimmen:    CN = PEEK(36866) AND 127                   (7)
```

LN = Anzahl der Zeilen:

```
Festlegen:    POKE 36867,(PEEK(36867) AND 1) OR LN*2     (8)
Bestimmen:    LN = (PEEK(36867) AND 126)/2               (9)
```

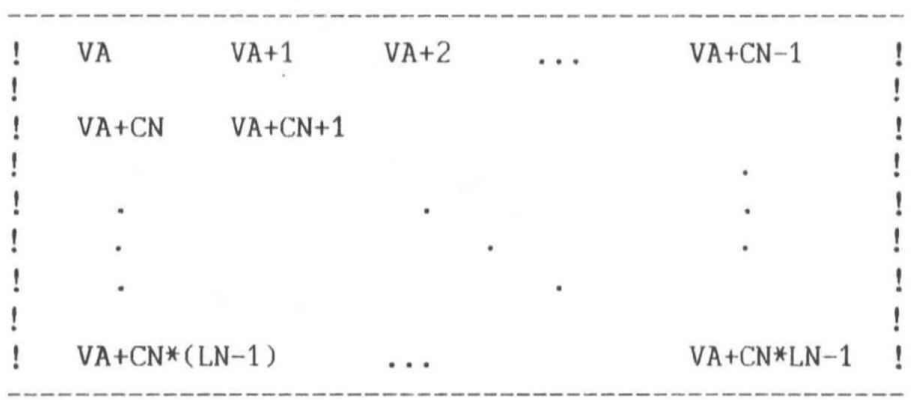


Bild 6.10.1

Bild 6.10.1 soll einen Bildschirm andeuten. Die Videospeicher-Zellen sind in der dargestellten Weise auf dem Bildschirm angeordnet. Dieses Anordnungsprinzip muß verstanden werden, um die spezielle Bit-Mapping-Speicherorganisation in geeigneter Weise durchführen zu können.

Jedes der CN*LN Videospeicherplätze enthalt einen Videocode-Wert (VD), naturlich einen Wert zwischen 0 und 255. Jeder dieser Werte muß logischerweise einen bestimmten Zeichengeneratorbereich von je 8 bzw. 16 Byte Lange, namlich die Pixelzeilen, adressieren, der den Videospeicherplatz somit in gewunschter Weise in Form eines sichtbaren Zeichens oder beliebigen Pixel-Rasters beim Bit-Mapping ausfüllt. Der Zusammenhang zwischen dem Videocode VD und den zugehorigen Pixelzeilenadressen PZA laßt sich in folgender Weise beschreiben:

Pixelzeilenadresse PZA:

Bestimmen: Pixelzeile	PZA =	
1	ZA + VD*PZN + 0	(10)
2	ZA + VD*PZN + 1	
.	.	
.	.	
8 bzw. 16	ZA + VD*PZN + 7 bzw. 15	

Pixelzeileninhalt PZI:

Festlegen: POKE PZA,PZI (11)

Bestimmen: PZI = PEEK(PZA) (12)

Der Pixelzeileninhalt gibt mit anderen Worten die Kombination der 8 Pixels einer Pixelzeile an.

BEISPIEL

Von der Theorie zur Praxis. Ziel unserer Überlegungen bei diesem Beispiel war, den Speicherbereich $1000-1FFF = #4096-8191 furs Bit-Mapping derart zu organisieren, daß Videospeicher und Zeichengenerator - im weiteren bezeichnen wir beide als Bit-Map-Speicher - in optimaler Weise zusammen darin Platz finden. Die Losung dieser Problemstellung finden Sie in Bild 6.10.2.

```
0 REM HIGH-RES
10 GOTO2000
500 SP=INT(X/8):AD=4352+SP*192+Y
510 POKEAD,PEEK(AD)OR(2^(SP*8-X+7))
520 RETURN
1000 POKE648,16
1010 POKE36866,20
1020 POKE36867,25
1030 POKE36869,204
1040 VD=16
1050 FORSP=0TO19
1060 FORZL=0TO11
1070 AD=ZL*20+SP
1080 POKEAD+4096,VD:VD=VD+1
1090 POKEAD+37888,6
1100 NEXTZL,SP
1110 FORBM=4352TO8191
1120 POKEBM,0:NEXT
1130 RETURN
2000 GOSUB1000
2100
2110
2120
3000 WAIT198,1:SYS52100
```

Bild 6.10.2

In der VC-20-Grundversion findet das Programm HIGH-RES in Bild 6.10.2 naturlich keinen Platz mehr. Bei den erweiterten Versionen andererseits reicht ein bloßes Laden nicht aus, weil der Bit-Map-Speicher sich mit dem Basic-Arbeitsspeicher überlappen würde.

In der >=8KByte-Version muß deswegen der Basic-Anfang vor dem Laden nach oben versetzt werden:

POKE 44,32:POKE 8192,0:NEW (Return)

In der 3KByte-Version reicht es aus, das Basic-Ende nach unten zu verschieben:

POKE 56,16:NEW (Return)

HIGH-RES enthält zwei universell einsetzbare Bit-Map-Unterprogramme:

Zeilen 500-520: In dem Punkt X,Y mit X=0...159 und Y=0...191 wird ein Pixel sichtbar gemacht.

Zeilen 1000-1130: Der Bit-Map-Speicher wird initialisiert und alle Pixels werden geloscht.

Erläuterungen zum Programm HIGH-RES:

Die POKE-Anweisungen in den Zeilen 1000, 1010 und 1030 enthalten in Kurzform das, was die Beziehungen in Tabelle A.5.6 sowie die Beziehung (6) fur folgende Werte herbeifuhren wurden:

ZA = 4096 VA = 4096 CN = 20

Die Zeile 1020 legt fest:

PZN = 16 LN = 12

Aus diesen Werten sind die folgenden ableitbar:

Insgesamte Zeichenzahl: CN*LN = 20*12 = 240
Videospeicher-Bereich: #4096-4335
Horizontale Auflosung: 20*8 = 160 Pixels
Vertikale Auflosung: 12*PZN = 12*16 = 192 Pixels
Gesamtauflosung: 160*192 = 30720 Pixels

Die Werte fur die Videocodes mussen einerseits in jeder Videospeicherzelle verschieden sein (unabhangige Pixelzuweisung), andererseits durfen sie keine Adressen des Videospeichers als Zeichengenerator-Bestandteil adressieren. Demzufolge darf VD nicht die Werte 0 bis 14 einnehmen (siehe Beziehung (10)). Wir lassen die 15 aus und beginnen bei 16 (Zeile 1040).

Die Schleife in den Zeilen 1050-1100 weist jeder der 240 Videospeicher-Zellen den Videocode VD zu, von 16 an aufwärts bis zu 255. PZA fängt somit bei #4352 an. Der letzte PZA-Wert ist 8191. Das Bild 6.10.3 verdeutlicht erganzend den Zusammenhang zwischen Videospeicher, VD und PZA.

Videospeicher-Adresse (#)	VD (#)	PZA-Bereich (#)
4096	16	4352 - 4367
4097	17	4368 - 4384
.	.	.
.	.	.
4335	255	8176 - 8191

Bild 6.10.3

Wir haben somit unser Ziel erreicht, einen zusammenhangenden Bit-Map-Speicher zu organisieren. Nur die 16 Speicherzellen #4336 - #4351 bleiben ungenutzt, was wohl verschmerzbar ist.

Die Programmzeile 1090 sorgt dafür, daß die zukünftig erzeugten Pixels auch wirklich zu sehen sind (Farbe "Blau").

Es folgt die Schleife 1110-1120, die in alle PZAs den Wert PZI=0 hineinPOKEd. Mit anderen Worten werden alle Pixels gelöscht, was somit die Funktion CLEAR SCREEN beim Bit-Mapping darstellt.

Die Zeilen 2100-2120 sind in HIGH-RES freigelassen und sollen die Programmteile für die jeweilige Berechnung der X,Y-Werte beinhalten. Beispiele dafür finden Sie im Bild 6.10.4. Beachten Sie, daß nach Berechnung von X und Y (deren Werte müssen unbedingt im oben angegebenen Bereich sein, sonst gibt's Programmsalat) der Sprung ins Unterprogramm 500 erfolgt, worin das eigentliche "Einschalten" der Pixels vorgenommen wird. Der Algorithmus für die Bitzuweisung in der betreffenden Videospeicherzelle ist recht einfach und bedarf keiner weiteren Erläuterung.

Lassen Sie sich nicht durch das SYS 52100 am Ende des Programms verwirren. Dies ist nur ein Sprung zu einer Adresse mit dem Inhalt 0, was die BREAK-Funktion des Mikroprozessors aktiviert. Die Konsequenz ist: Umschalten von Bit-Mapping in VC-20-"Mapping".

```
0 REM DEMO: PARABEL
2100 FORX=0TO159
2110 Y=191*(1-(X/80-1)↑2)
2120 GOSUB500
2130 NEXT
```

```
0 REM DEMO: GERADEN
2100 X=X+1:IFX>159THENX=0
2110 Y=Y+1:IFY>191THENY=0
2120 GOSUB500
2130 GOTO2100
```

```
0 REM DEMO: RASTER
2100 FORX=0TO159
2110 FORY=XAND1TO191STEP2
2120 GOSUB500
2130 NEXTY,X
```

```
0 REM DEMO: KREISE
2100 FORI=1TO4
2110 H1=I*20:FORX=80-H1TO80+H1
2120 IFX>159THEN2160
2130 HV=SQR((I/4)↑2-(X/80-1)↑2)
2140 Y=96*(1+HV):GOSUB500
2150 Y=96*(1-HV):GOSUB500
2160 NEXTX,I
```

```
0 REM DEMO: ZUFALLSPUNKTE
2100 X=160*RND(1)
2110 Y=192*RND(1)
2120 GOSUB500
2130 GOTO2100
```

Bild 6.10.4

6.11 ASCII/VIDEOCODE-KONVERTIERUNG

Im VC-20-Handbuch finden Sie auf den Seiten 141-142 die Bildschirm-Codes (sie werden auch als Videocodes bezeichnet) und auf den Seiten 145-147 die ASCII-Codes. In beiden Tabellen finden wir die gleichen Zeichen, nur mit teilweise anderen Code-Werten. In manchen Programmen ist es vonnoten, per Konvertierungsbeziehung diese beiden Codes gegenseitig umzurechnen.

```
0 REM VIDEO-ASCII-CODEUMWANDLUNG
100 INPUTVI
110 GOSUB1000
120 PRINTAS
130 GOTO100
1000 VI=VIAND127
1010 IFVIAND32THEN1040
1020 IFVIAND64THENAS=VIOR32:RETURN
1030 AS=VIOR64:RETURN
1040 IFVIAND64THENAS=VIAND63OR128:RETURN
1050 AS=VI:RETURN
```

Bild 6.11.1

Die Programme in den Bildern 6.11.1 und 6.11.2, insbesondere deren Unterprogramme ab Zeile 1000 leisten diese Konvertierung. Überprüfen Sie sie mal! Zu Demonstrationszwecken sind einige Zeilen vorangestellt, mittels denen Sie im Falle der ASCII-Videocode-Umwandlung einen ASCII-Code (Variable AS) eingeben können, z.B. 65 für "A", und den korrelierenden Videocode-Wert (Variable VI), im unserem Beispiel: 1, zuruckerhalten.

```
0 REM ASCII-VIDEO-CODEUMWANDLUNG
100 INPUTAS
110 GOSUB1000
120 PRINTVI
130 GOTO100
1000 IFASAND128THENVI=ASAND127OR64:RETURN
1010 IFNOTASAND64THENVI=AS:RETURN
1020 IFASAND32THENVI=ASAND95:RETURN
1030 VI=ASAND63:RETURN
```

Bild 6.11.2

6.12 8-FACHE VERGRÖSSERUNG DER VC-20-ZEICHEN

Wenn wir uns die Zeichen des VC-20 auf dem Fernsehbildschirm anschauen, halten wir es nicht fur moglich, wie sehr uns unsere Augen dabei betrugen. Unser Auge fullt nämlich die an sich groben Zeichenpixelraster derart aus, daß wir keine Unfeinheiten mehr wahrnehmen. Uberzeugen Sie sich davon! Laden Sie das Programm in Bild 6.12.1, starten es und geben ein beliebiges Zeichen ein. Jedes einzelne Pixel wird daraufhin in Zeichengroße und in der richtigen Anordnung dargestellt. Daruberhinaus wird eine horizontale und vertikale Skala in den Bereichen 0...7 angezeigt, so daß genau erkannt werden kann, an welcher Stelle der jeweilige Pixelpunkt sich befindet.

Das Programm ist 319 Bytes lang und lauft in jeder VC-20-Version. Das eingegebene Zeichen ist das dritte Zeichen auf dem Bildschirm, dessen Adresse Z2 in Zeile 20 berechnet wird. In Zeile 60 erhalt B den Inhalt (den Video-Code also) der Adresse Z2, und der Variablen A wird diejenige in den Zeichengeneratorbereich fallende Adresse zugewiesen, ab der in 8 Bytes die 8 einzelnen Pixel-

zeilen des eingegebenen Zeichens gespeichert sind. Die restlichen Anweisungen sorgen für die Darstellung auf dem Bildschirm.

```
10 REM 8FACHE VERGROESSERUNG ALLER ZEICHEN
20 Z2=PEEK(648)*256+2
30 FORCL=0TO7:BI(CL)=2↑(7-CL):NEXT
40 PRINTCHR$(147)
50 PRINTCHR$(19);:INPUTA$
60 B=PEEK(Z2):A=B*8+32768+(PEEK(36869)AND2)*1024
70 FORI=1TO12:POKE781,I:SYS60045:NEXT
80 FORCL=0TO7:POKE211,0:PRINTCHR$(17)CL;
90 FORCP=0TO7
100 IFPEEK(A+CL)ANDBI(CP)THENPRINTCHR$(18);
110 PRINT" "CHR$(146);
120 NEXTCP,CL
130 PRINT:PRINT:PRINTSPC(3)"76543210"
140 GOTO50
```

Bild 6.12.1

6.13 ZEICHEN IN DOPPELTER HÖHE

Das Programm in Bild 6.13.1 ist sowohl ein Demonstrationsbeispiel für die Verwendung der Tabelle A.5.8 im Anhang A.5 als auch eine nützliche prinzipiell in jedem Programm verwendbare Einrichtung, welche die Darstellung der VC-20-Zeichen in doppelter Höhe erlaubt. Das Programm läuft zwar in jeder VC-20-Version, belegt jedoch den 1536 Bytes umfassenden RAM-Bereich $1800-1DFF bzw. #6144-7679 zur Speicherung des hierbei "selbstgestrickten" Zeichengenerators für die Zeichen des Videocodes #0-95 gemäß Tabelle auf den Seiten 141-142 im VC-20-Handbuch. Falls also größere Programme in der VC-20-Version mit >=8KByte-Speichererweiterung zum Einsatz kommen sollten, ist es empfehlenswert, den Basic-Anfang-Vektor mittels des im Kap. 2.1 beschriebenen Verfahrens auf $1E01 bzw. #7681 zu setzen. In diesem Fall ist es allerdings nötig, das "POKE 56,24" aus Zeile 20 herauszunehmen.

Die Variable AD erhält die Anfangsadresse #6144 für den nachfolgend generierten Zeichengenerator. Das POKE 657,128 unterbindet die Zeichenumschaltemöglichkeit, denn aus Gründen der Speicher-

platzeinsparung ist die Verdoppelung nicht für alle Zeichen vorgesehen. Damit überhaupt ein Zeichen in 16 einzelnen Pixel-Zeilen auf dem Bildschirm dargestellt werden, ist das Setzen den Double-Bits (D-Bit = Bit 0 in Adresse $9003, s. Tabelle A.5.10) per POKE 36867,27 erforderlich. Zeile 40 spiegelt die in Tabelle A.5.8 gezeigte Regel wider. In den Zeilen 60-70 werden 2mal hintereinander die gleichen Pixel-Zeilen abgespeichert, was schließlich die doppelte Höhe ausmacht.

```
10 REM  DOPPELTE HÖHE
20 POKE 56,24:CLR:AD=6144
30 POKE 657,128:POKE 36867,27
40 POKE 36869,PEEK(36869) AND 24 OR 14
50 FOR  I=0 TO 767
60 POKE 2*I+AD,PEEK(32768+I)
70 POKE 2*I+AD+1,PEEK(32768+I)
80 NEXT
```

Bild 6.13.1

6.14 UMLAUTE UND ß

Das in Bild 6.14.1 gezeigte Programm UMLAUTE UND SCHARFES S ist ein zusätzliches Demonstrationsprogramm, welches als ein weiteres Beispiel für die Erstellung eines eigenen Zeichengenerators angesehen werden kann. Es läuft in allen VC-20-Versionen und soll an sich nur ein Anstoß dafür geben, in anderen Programmen, eventuell Textverarbeitungsprogrammen, die Darstellungsmoglichkeit für die gewohnten Umlaute vorzusehen.

In Zeile 50 wird das Basic-Ende auf den Anfang (Adresse #6144) des neuen Zeichengenerators gesetzt.

In Zeile 60 wird die Zeichensatzumschaltemöglichkeit ausgeschaltet. Das Umschalten hätte nämlich wenig Zweck, weil der neue Zeichengenerator längst nicht so groß ist (1535 Byte), daß er zwei Zeichensätze unterzubringen vermag.

```
10 REM UMLAUTE UND SCHARFES S
20 PRINTCHR$(147)"FOLGENDE #ZEICHEN WERDEN"
30 PRINT"DEN VERÄNDERT:"
40 PRINT:PRINT"[ ] \ @ ^ £ ▼"
50 POKE56,24:CLR:AD=6144
60 POKE657,128
70 POKE36869,PEEK(36869)AND240OR14
80 FORI=0TO1535
90 POKEI+AD,PEEK(34816+I):NEXT
100 DATA24,36,68,120,68,68,120,64
110 FORI=27*8+ADTOI+7
120 READA:POKEI,A:NEXT
130 J=29*8+AD:POKEJ,36
140 FORI=1TO7
150 POKEJ+I,PEEK(AD+8+I):NEXT
160 J=64*8+AD:POKEJ,30
170 FORI=1TO7
180 POKEJ+I,PEEK(AD+8*65+I):NEXT
190 POKEAD,36
200 FORI=1TO7
210 POKEAD+I,PEEK(AD+8*15+I):NEXT
220 J=122*8+AD:POKEJ,90
230 FORI=1TO7
240 POKEJ+I,PEEK(AD+8*91+I):NEXT
250 J=28*8+AD:POKEJ,36
260 FORI=1TO7
270 POKEJ+I,PEEK(AD+8*21+I):NEXT
280 J=105*8+AD:POKEJ,36
290 FORI=1TO7
300 POKEJ+I,PEEK(AD+8*85+I):NEXT
```

Bild 6.14.1

In Zeile 90 wird gemaß Regel in Tabelle A.5.6 im Anhang A.5 der Zeichengenerator-Anfang auf die Adresse #6144 festgelegt, worauf in den Zeilen 80-90 die ersten 1536 Pixelzeilen des VC-20-Zeichensatzes 2 ($8800-8FFF = #34816-36863) in den neuen Zeichengenerator-Bereich kopiert werden.

Da die Videocodes (s. Kap. 6.10) nun Pixelzeilen ab Adresse #6144 adressieren, und diese ihrerseits sich jetzt im RAM-Bereich befinden, lassen sich die Buchstaben und Zeichen nach Belieben und eigenem "Geschmack" gestalten. Auch gotische, altdeutsche, kyrillische oder griechische Schrift ist natürlich kreierbar. Bleiben wir aber bei den Umlauten.

Wie Sie den weiteren Zeilen entnehmen konnen, ist gar nicht so viel zu tun, um das Ziel "Umlaute und ß" zu realisieren. Naturlich nutzen wir hierbei die Tatsache aus, daß die Pixelzeilen der

Buchstaben A, O, U, a, o und u bereits vorhanden sind, und brauchen somit nur noch die Punkte in geeigneter Weise darüber setzen. Das "ß" muß allerdings von vornherein selbst gestaltet werden (Pixelzeilen-Werte in der DATA-Zeile 100).

Durch die Hinzunahme der Umlaute und dem "ß" sind uns einige Zeichen verlorengegangen. Mit anderen Worten, folgende Tabelle zeigt Ihnen, mit welchen Tasten Sie die neuen Zeichen auf den Bildschirm bringen:

Taste:	ergibt Zeichen:
[	ß
]	ä
SHIFT + "*":	Ä
@	ö
SHIFT + "@":	Ö
"Engl. Pfund":	ü
SHIFT + "Engl. Pfund":	Ü

Anhang

A.1 ABKÜRZUNGEN

A	Akku-Register
A-Bereich	Speicherbereich $A000-AFFF = #40960-45055
AV	Ausbauversion: GV + Speichererweiterung um mindestens 8K Byte
B	Break-Flag
BAM	**B**lock **A**vailable **M**ap
Bd	Baud
Bu	Buchstaben
B-Bereich	Speicherbereich $B000-BFFF = #45056-49151
B-PGM	Basic-Programm
B-UP	Basic-Unterprogramm
B6	Bit 6
B7	Bit 7
C	Carry-Flag
CLR HOME	Bildschirm löschen und Cursor zur 1. Bildschirmzeichenstelle bringen
CR	**C**arrige **R**eturn (Wagenrucklauf)
CRSR	Cursor
CRSR DWN	Cursor nach unten (down)
CRSR RI	Cursor nach rechts (right)
CRSR UP	Cursor nach oben

CR0 - CRF	Control Register des VIC
D	Dezimal-Flag
DEZ	Dezimal
DOS	Drive Operating System; das Betriebssystem, welches die Floppy unterstutzt
GND	engl.: ground, = Masse (Bezugspotential)
GV	Grundversion (ohne jegliche Speichererweiterung
H	siehe HIGH
HEX	Hexadezimal
HIGH	HIGH part of address, das wertmaßig hohere Byte der aus 2 Bytes bestehenden Adresse Beispiel: in $AB32 ist $AB der HIGH-Part
HOME	Cursor wird zur 1. Bildschirmzeichenstelle gebracht
I	Interrupt-Disable-Flag
IRQ	siehe unter "Interrupt-Routine" im Anhang B: Begriffserklarungen
I/O	Input/Output = Ein-/Ausgabe
Kap.	Kapitel
L	siehe LOW
LOW	LOW part of address, das wertmaßig niedere Byte der aus 2 Bytes bestehenden Adresse Beispiel: in $AB32 ist $32 der LOW-Part

M-Befehle	Maschinensprache-Befehle, zumeist in Mnemonics angegeben, wie z.B.: LDA #$44 = Lade Register A (Akku) mit dem Wert $44
M-PGM	Maschinensprache-Programm (beim VC-20 für den Mikroprozessortyp 6502)
M-UP	Maschinensprache-Unterprogramm
N	Negativ-Flag
NMI	Non Maskable Interrupt; siehe unter "Interrupt-Routine" im Anhang A.2: Begriffserklärungen
PGM	Programm
PSW	Programm-Status-Wort; siehe Begriffserklärung in Anhang A.2
RS 232	siehe unter "V.24-Schnittstelle" im Anhang B: Begriffserklarungen
RTTY	Funkfernschreiben
RVS	REVERS = Zeicheninvertierung; ON=Ein, OFF=Aus
s.	siehe
S.	Seite
ST	Status
Tab.	Tabelle
UP	Unterprogramm; steht auch in Verbindung mit CRSR UP (Cursor nach oben)
V	Overflow-Flag
VIA	Versatile Interface Adapter (6522); siehe weiter Erlauterung in Anhang A.2

VIC	**V**ideo **I**nterface **C**hip (6561); siehe weitere Erläuterungen in Anhang A.2
V.24	siehe unter "V.24-Schnittstelle" im Anhang A.2: Begriffserklärungen
WR	Wagenrücklauf
Z	Zero-Flag
Zi	Ziffern
ZL	Zeilenvorschub
#	steht vor einer Dezimalzahl zwecks Unterscheidung von einer Hexadezimalzahl
#$	steht vor einer Hexadezimalzahl, um diese als Inhalt zu kennzeichnen (sie konnte ja auch die Bedeutung einer Adresse haben)
#$KK	heißt: beliebige Konstante als Inhalt eines Bytes
$	steht vor einer Hexadezimalzahl zwecks Unterscheidung von einer Dezimalzahl
$HH	das höherwertige Byte einer Adresse (H=HIGH)
$HHLL	bezeichnet eine beliebige Adresse; HH steht für höherwertiges Byte (H=HIGH), LL steht für niederwertiges Byte (L=LOW)
$LL	das niederwertige Byte einer Adresse (L=LOW)
$ZZ	Zero-Page-Adresse; $ZZ entspricht in der Notation $HHLL der Adresse $00ZZ
3K-V	GV + 3K-Speichererweiterung ($0400-0FFF)
8K-V	GV + 8K-Speichererweiterung ($2000-3FFF)

16K-V	GV + 16K-Speichererweiterung ($2000-5FFF)
24K-V	GV + 24K-Speichererweiterung ($2000-7FFF)

A.2 BEGRIFFSERKLÄRUNGEN

A/D-Wandler

Als A/D-Wandler bezeichnet man ein Bauteil, das eine **a**naloge Größe in eine **d**igitale Größe **umwandelt** (konvertiert). Der VIC 6561 (Video-Interface-Chip) im VC-20 besitzt zwei integrierte A/D-Wandler, die analoge Spannungsgroßen zwischen 0 Volt und 5 Volt in digitale Großen zwischen #255 und 0 umwandelt. Die digitalisierten Großen können in den Kontrollregistern CR8 (Adresse $9008=#36872) und CR9 (Adresse $9009=#36873) des VIC abgelesen werden.

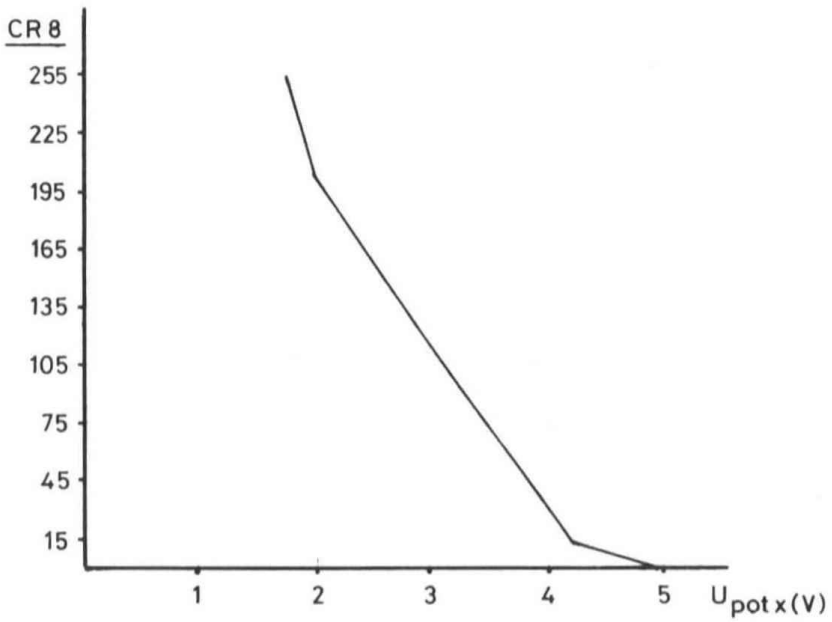


Bild A.2.1: A/D-Wandler

Die graphische Darstellung (Bild A.2.1) zeigt, daß eine lineare Umwandlung nur zwischen 2 Volt und 4 Volt gegeben ist. Der Programmierer sollte das beachten und in seinem Programm berücksichtigen oder aber die Hardware entsprechend dimensionieren.

Basic-Loader

ist ein Basic-Programm, welches im wesentlichen dazu dient, ein darin enthaltenes Maschinensprache-Programm in einem zumeist vom Benutzer frei wählbaren Speicherbereich zu generieren (s. Kap. 5.2 "Ein spezieller Basic-Loader").

Basic-Token siehe Token

Baudot-Code

international festgelegter 5-Bit-Code zur seriellen Datenübertragung (s. Bild A.2.2). Als sogenanntes Start-Bit wird eine logische 0 gesendet, die den Empfänger veranlaßt, das folgende 5-Bit-Datenwort zu empfangen. Nach dem Datenwort muß mindestens 1.5 Bit lang eine logische 1 als Pause zwischen zwei Datenworten gesendet werden.

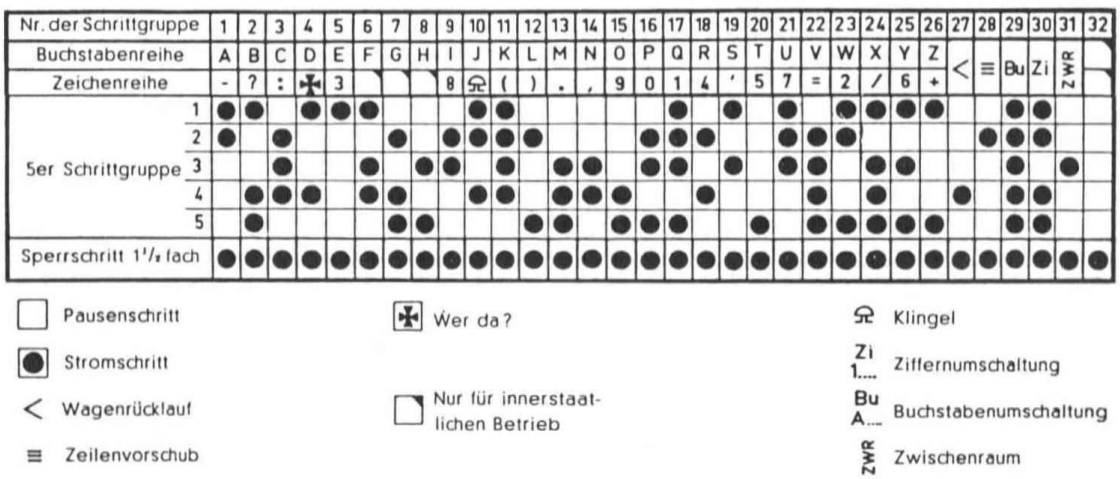


Bild A.2.2

Mechanische Fernschreiber nutzen diese Pause, um das empfangene Zeichen abzudrucken oder Befehle wie:

WR = Wagenrücklauf
ZL = Zeilenvorschub
Bu = Umschaltung in Buchstabenebene
Zi = Umschaltung in Zifferneben e etc.

auszuführen. Eine Paritatsprufung ist nicht vorgesehen.

Bildschirm-Code

Die Inhalte im Video-Speicher werden mittels des Bildschirm-Codes gespeichert (s. Anhang H im VC-20-Handbuch). Z.B. wird ein großes A im Video-Speicher mit dem Wert 1 codiert. Ist das A dagegen der Inhalt eines Strings oder einer Stringvariablen, so wird es mittels ASCII-Code mit dem Wert 65 (s. Anhang J im VC-20-Handbuch) codiert. Hilfs-Programme zur gegenseitigen Transformierung beider Codes finden Sie in Kap. 6.11 "ASCII-/Videocode-Konvertierung".

Bildschirm-Speicher s. Video-Speicher

Bubble-Sort

Als Bubble-Sort bezeichnet man ein Sortierverfahren, bei dem immer zwei benachbarte Elemente einer Liste miteinander verglichen und gegebenenfalls ausgetauscht werden. Der Bubble-Sort-Algorithmus muß so lange durchlaufen werden, bis alle Elemente der Liste in der richtigen Reihenfolge stehen.

Vorteil: Programme nach dem Bubble-Sort-Verfahren sind einfach zu erstellen.
Nachteil: Das Verhaltnis von Große der Liste zur Laufzeit des Programms andert sich quadratisch. Das heißt, fur die doppelte Menge von Daten benotigt das PGM die vierfache Zeit.

Default

ein verdeutschter englischer Ausdruck (wie so viele!), was so gut wie "Standardwert" oder "Ausgangswert, falls sonst nichts angegeben ist" heißt; als einfaches Beispiel kann man hierbei die Aus-

gangswerte, also Default-Werte, von Basic-PGM-Variablen nennen: falls nichts von vornherein festgelegt worden ist, erhalten die numerischen Variablen den Wert 0 und die Stringvariablen die leere Zeichenkette "".

Deskriptor

Ein String bzw. eine Zeichenkette wird u.a. mithilfe von 3 Bytes identifiziert, welche die Länge und die absolute Anfangsadresse der Zeichenkette festlegen:

1 Byte für die Länge, 2 Bytes für die Adresse (L,H)

Da also nur 1 Byte für die Länge vorgesehen ist, dürfen Zeichenketten, welche mithilfe von Deskriptoren beschrieben werden (die in Basic benutzten String-Variablen beispielsweise), nicht länger als 255 Zeichen lang sein.

Directory Verzeichnis

Dump Speicherinhalt-Liste

Expansion-Port

wird im VC-20-Handbuch als "Speichererweiterungsanschluß" und als "Hauptspeicher-Erweiterung" (S. 149, 150) bezeichnet. Die Belegung dieses Ports bei Draufsicht von außen auf den VC-20 wird auf S. 150 (VC-20-Handbuch) gezeigt.

Farbspeicher

ist derjenige Speicherbereich, in dem die Farben jedes Zeichens des Video-Speichers eingetragen sind.
Für die Versionen GV, sowie ab 8K-V sind das die Bereiche:

GV und 3K-V:	$9600 - 97F9	#38400 - 38905
ab 8K-V:	$9400 - 95F9	#37888 - 38393

Eine Berechnungsformel, die den unterschiedlichen Bereichszuweisungen abhängig vom jeweiligen Speicherausbau Rechnung trägt, wird in Kap. 6.3 "Lösung des Portabilitäts-Problems" angegeben.
Nur 4 Bits der unteren Bytehälfte werden als Inhalt eingetragen. Die Farbwerte entsprechen denen auf der Tastatur angegebenen abzüglich 1, z.B. ist hier Blau=6 und Schwarz=0. Die oberen 4 Bits weisen Zufallswerte auf, so daß diese - sofern man die korrekten Werte des Farbspeichers in einem Algorithmus verwenden mochte - mittels AND 15 weggefiltert werden mussen.
Will man also z.B. den korrekten Farbwert des 1. Zeichens auf dem Bildschirm bei einem VC-20 in GV ermitteln - #38400 ist die Farbspeicherzelle zur Video-Speicherzelle #7680, #38401 zu #7681 etc. - so geschieht dies einfach durch: PEEK(38400) AND 15.

File Datei

Fließkomma-Zahl s. Gleitkomma-Zahl

Gleitkomma-Zahl

auch als Fließkomma-Zahl bezeichnet; beim VC-20 eine mittels 5 Bytes festgelegte Zahl zwischen -1.70141*10^38 und +1.70141*10^38. An sich umfassen 5 Bytes lediglich einen Zahlenraum von 2^(5*8) = 2^40 = 1.0995*10^12. Dies hieße jedoch, man hätte eine Genauigkeit von 11 Stellen. Auf diesen Stellengenauigkeitsanspruch verzichtet man hier jedoch und knappst einige Bits von den 40 zur Verfügung stehenden ab, um den Exponenten in einem größeren Bereich variieren lassen zu konnen (-38 bis +38).
An dieser Stelle sollte auch erwahnt werden, daß Rundungsfehler bei Addition bzw. Subtraktion auftreten, die sich insbesondere bei Berechnungen in Schleifen zu größeren Resultatungenauigkeiten aufschaukeln können. Prüfen Sie es selbst nach, indem Sie eingeben: PRINT 100-100.1 (Return).
Als Ergebnis erhält man -.099999994.

Hexadezimalzahl

Dezimalziffern konnen Werte von 0 bis 9 annehmen, Hexadezimalziffern dagegen Werte von 0 bis 15; um jedoch die zweistellige Darstellung der Ziffern zwischen 10 und 15 zu vermeiden, ordnet man diesen die Buchstaben A bis F zu; woran man sich beim Umgang mit Hexadezimalzahlen gewohnen muß, ist, daß im Gegensatz zu Dezimalzahlen nicht die 2. Stelle mit 10, die 3. Stelle mit 100 etc. sondern die 2. Stelle mit 16, die 3. Stelle mit 256 und die 4. Stelle mit 4096 gewichtet werden.

HIGH-Byte

hoherwertiges Byte; z.B. ist in der Adresse $12AB das HIGH-Byte: $12

HOME Cursor zur 1. Bildschirmzeichenstelle

Integer-Zahl

an sich jede ganze Zahl, hier beim VC-20 speziell Zahlen zwischen -32768 und +32767.
Der Zahlenraum ist somit 65536 = 2^8 * 2^8 und benotigt nur 2 Bytes zur eindeutigen Identifizierung jeder Zahl in diesem Bereich.

Interface

Ein verdeutschter englischer Ausdruck; ubersetzt heißt er zwar "Schnittstelle", hat aber nicht die gleiche Bedeutung wie dieser Ausdruck. Denn beim Interface handelt es sich nicht selten um eine Hardware-Einrichtung, die zwei verschiedene Schnittstellen (siehe "Schnittstelle") funktionsfahig verbindet.

interrupt gesteuert

Die zwei im VC-20 eingebauten I/O-Bauelemente, VIA 6522, weisen mannigfaltige (VIA = Versatile Interface Adapter) Moglichkeiten

der Ein-/Ausgabesteuerung auf. U.a. hat jeder von ihnen zwei Timer und spezielle Register, welche nach entsprechender Programmierung den Sprung zur Interrupt-Routine erzwingen können, wenn die Timer abgelaufen sind oder vorher festgelegte Signale (z.B. Anstieg von "0" auf "1") auf den Steuerleitungen anfallen. Programme, die sich dieser Möglichkeiten bedienen, nennt man interrupt gesteuert.

Interrupt-Routine

Routinen, welche nach anfallender Interrupt-Anforderung von selbst angesprungen werden. Beim Mikroprozessor des VC-20, dem 6502, gibt es 3 Hierarchiestufen von Interrupt-Anforderungen:

		Einsprungadresse:
hochste Prioritat:	RESET	$FFFC,$FFFD
mittlere Prioritat:	NMI	$FFFA,$FFFB
unterste Prioritat:	IRQ	$FFFE,$FFFF

Während RESET und NMI (Non-Maskable Interrupt) sich per M-PGM nicht verhindern lassen, gibt es für IRQ die M-Befehle SEI, CLI, PLP zur EIN/AUS-Schaltung der IRQ-Verhinderung.

Speziell auf den VC-20 bezogen:
RESET läßt sich hardwaremaßig einfach durch Kurzschließen irgendeiner RESET-Leitung mit einer GND-Leitung erzeugen oder aber softwaremaßig durch Sprung in die in den Adressen $FFFC,$FFFD (L,H) angegebene Adresse: $FD22 = #64802
Die Eingabe von SYS 64802 entspricht also dem Einschalten des VC-20 mit dem Unterschied, daß die vorher gespeicherten Daten im RAM-Arbeitsspeicher erhalten bleiben (bis auf 2 Bytes, s. Kap. 4.5 "VC-OLD: NEW rückgangig machen").
Da allerdings der Programmierer ab und zu wegen totaler PGM-Aufhängung gar nicht mehr den Befehl SYS 64802 einzugeben in der Lage ist, empfiehlt es sich, einen RESET-Schalter anzubringen (s. unter "RESET").

NMI springt auf diesbezügliche Anforderung in eine Routine ab in $FFFA,$FFFD angegebene Adresse $FEA9, von wo ab per M-Befehl SEI der IRQ ausgeschaltet wird und in die in $0318, $0319 stehende Adresse $FEAD gesprungen wird. Der letztgenannnte vektorisierte

Sprung laßt sich vom Programmierer umleiten, weil die Zellen $0318,$0319 zum RAM-Bereich gehoren.

IRQ ist der bei den Maschinensprache-Programmierern wohl am haufigsten verwendete Interrupt, weil er nicht nur per M-Befehle SEI, CLI, PLP steuerbar ist, sondern auch fur die I/O-Steuerung eingesetzt wird. Die Einsprungadresse der IRQ-Routine befindet sich in $FFFE,$FFFF: $FF72. Ab $FF72 werden zusatzlich A,X,Y gerettet (von selbst wird namlich schon die Rucksprungadresse und das Statusregister auf den Stack gelegt) und das BRK-Flag des geretteten Statuswortes uberprüft.
Ist das BRK-Flag gesetzt, wird zur in $0316, $0317 stehenden Adresse, also $FED2 gesprungen, im anderen Fall zur in $0314, $0315 stehenden Adresse $EABF. Beide Vektoren lassen sich andern, weil sie im RAM-Bereich liegen (s. Anhang A.3 unter $0314).

Keller RAM-Speicherbereich $0100-$01FF

Link

Vektor zum Zweck einer Verbindung bzw. Verkettung von logisch zusammengehorigen Daten, z.B. die ersten 2 Bytes einer Basic-Zeile, welche ein Link zum Anfang der Basic-Folgezeile darstellen.

LOW-Byte

niederwertiges Byte; z.B. ist in der Adresse $12AB das LOW-Byte: $AB

Offset

heißt so viel wie Differenzwert in Verbindung mit einem Bezugswert.
Beispiel: die Adresse $2014 sei der Bezugswert, so ist Y=5 der Offset fur die Adresse $2019, wenn man die Adresse $2019 ausdruckt durch $2014+Y.

Page

Ein 256 Bytes umfassender Speicherbereich von $HH00-HHFF (HH ist ein beliebiges höhere Byte). Zum Beispiel nimmt die Page $21 den Speicherbereich $2100-21FF ein.

Pixel

Ein Buchstabe, eine Zahl oder ein Graphikzeichen wird auf dem Bildschirm durch eine Zusammenstellung von einzelnen Punkten, den Pixels, dargestellt. Ebenso druckt ein sogenannter Matrixdrucker einzelne Pixels aus, um Zeichen auf Papier darzustellen.

Pointer s. Vektor

Programm-Status-Wort

ist das Register eines Mikroprozessors, welches sämtliche Flags enthält; beim VC-20-Prozessor 6502 sind dies:

Bit 0: Carry-Flag (C)
Bit 1: Zero-Flag (Z)
Bit 2: Interrupt-Disable-Flag (I)
Bit 3: Dezimal-Flag (D)
Bit 4: Break-Flag (B)
Bit 5: bleibt unbenutzt
Bit 6: Overflow-Flag (V)
Bit 7: Negativ-Flag (N)

Z.B. wird das Zero-Flag gesetzt, wenn nach einer Rechenoperation im Akku-Register das Ergebnis $00 steht. Der jeweilige Status der Flags wird zur Realisierung von M-Programmverzweigungen benutzt. Beim Befehl BNE (**B**ranch, if **N**ot **E**qual Zero) z.B. wird nur dann gesprungen, wenn das Zero-Flag nicht gesetzt ist.

relocatable

oft benutztes Wort in Verbindung mit M-PGM'en und heißt: frei verschiebbar.
Ein M-PGM ist relocatable, wenn es nicht vom absoluten Standort im Speicher abhangig ist. Es sind darin also z.B. keine absoluten Sprunge vorhanden.

RESET

ist eine wichtige Mikroprozessor-Funktion, bei der der Mikroprozessor in einen definierten und funktionsfahigen Ausgangszustand (zurück-)versetzt wird. Beim Einschalten der Mikroprozessor-Stromversorgung tritt also RESET auf jeden Fall in Aktion. Die hardwaremäßige Initialisierung des Mikroprozessors ist aber nicht die einzige Aufgabe von RESET, denn damit wurde noch lange nicht ein kompliziertes Gebilde wie der VC-20 (dessen Herzstuck zwar der Mikroprozessor ist) zum Funktionieren gebracht werden. Deswegen erfullt RESET noch eine weiter entscheidene Aufgabe. Es wird nämlich gleich im Anschluß an die hardwaremäßige Initialisierung ein Sprungbefehl ausgeführt, ein Sprung zu einer vom Hersteller des Mikroprozessors festgelegten Adresse. Speziell beim VC-20-Mikroprozessor 6502 ist dieser Sprung ein indirekter zu einer Adresse, welche in den Adressen $FFFC, $FFFD (L,H) steht.
Beim VC-20 ist die Einsprungadresse $FD22 = #64802. Sie initialisert eine großere Anzahl von Zero-Page-Werten, Links, Vektoren etc. und programmiert die I/O-Bausteine VIA 6522 derart, daß ein vernünftiger Betrieb von vornherein moglich ist (z.B. muß ja die Tastaturbetatigung fur den VC-20 "verstandlich" sein).
Die RESET-Routine kann hardwaremäßig auch durch Kurzschließen von einem RESET-Ausgang und GND (Masse) oder softwaremäßig durch Eingabe von SYS 64802 angesprungen werden. Da letzteres nur dann moglich ist, wenn der VC 20 uberhaupt noch uber die Tastatur etwas annehmen kann (was ja beim "Absturz" des Systems in den seltensten Fallen noch drin ist), empfiehlt sich grundsatzlich der hardwaremäßige Einsprung, z.B. mittels eines Tastschalters zwischen:

einem der RESET-Pins:
Pin X der Hauptspeicher-Erweiterung (s. S. 150 im Handbuch)

od. Pin 6 der seriellen Ein-/Ausgabe (s. S. 151 im Handbuch)
od. Pin 3 des User-Ports (Anwender Ein-/Ausgabe) (s. S. 152 im Handbuch)

und einem der GND-Pins:

Pin A der Hauptspeicher-Erweiterung
od. Pin Z " " "
od. Pin 1 " " "
od. Pin 22 " " "
od. Pin 2 des Audio/Video-Ausgangs (s. S. 151 im Handbuch)
od. Pin 2 der seriellen Ein-/Ausgabe
od. Pin A des Datassettenports (s. S. 151 im Handbuch)
od. Pin 1 " "
od. Pin A des User-Ports
od. Pin N " " "
od. Pin 1 " " "
od. Pin 12 " " "

Vorsicht! Pin 11 des User-Ports ist **kein** GND-Pin (Druckfehler im Handbuch!).
Eine sinnvolle Utility in Verbindung mit RESET ist das Programm VC-OLD (s. Kap. 4.5), mittels der es moglich ist, ein Basic-PGM auch nach dessen Absturz zu retten (falls nicht versehentlich in das PGM selbst hineingePOKEd wurde).

Routine zumeist als UP verwendetes PGM

RS232-Schnittstelle siehe V.24-Schnittstelle

Scanner-Routine siehe Anhang A.3, Adr. $0073-008A

Schnittstelle

Ein gedachter oder wirklich realisierter Schnitt (z.B. in Form eines Steckers) zumeist in einer Datenubertragungsstrecke. Zu einer ausreichenden Schnittstellenbeschreibung gehort die exakte Erklarung uber die Anzahl der Leitungen, deren Spannungen/Strome und deren Signalbedeutungen an dieser "aufgeschnittenen" Stelle.

SPACE Leerzeichen (-taste)

Stack

speziell beim VC-20 (6502-Prozessor): RAM-Speicherbereich $0100-$01FF

Stapel siehe Stack

String Zeichenkette

String-Variable

Variable, der eine Zeichenkette zugeordnet wird.

Timer

Zeitüberwachung; es ist eine softwaremäßig realisierte "Eieruhr". Man stellt sie auf eine bestimmte Zeit ein. Wenn sie abgelaufen ist, geht es im PGM in einer vorher festgelegten Weise weiter. Ein typischer Basic-Timer ist z.B.:

```
FOR I=1 TO N*1000:NEXT
```

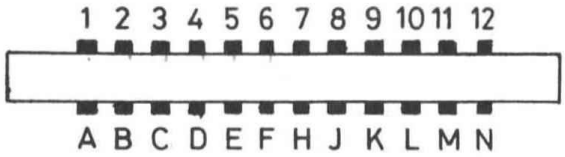
um das PGM an dieser Stelle ca. N Sekunden verweilen zu lassen.

Token

ein Basic-Befehl, z.B. PRINT wird nach Eingabe nicht in Form von P_R_I_N_T, also mit 5 Zeichen, abgespeichert, sondern nur mit einem Zeichen, bei PRINT mit $99=#153. Dieses eine Byte zur Indentifizierung des Befehls wird als Token bezeichnet. Das Verfahren spart Speicherplatz ein und vergrößert überdies die Geschwindigkeit der Basic-Programme. Eine Liste aller Tokens befindet sich im Programmier-Handbuch S. 178.

USER-Port

Der USER-Port ist eines der "Tore" des VC-20 zur Außenwelt. Da im VC-20-Handbuch eine teils falsche teils unvollstandige Aufstellung dessen Pins und Bedeutungen gezeigt wird, soll an dieser Stelle mittels Bild A.2.3 eine weitere eingebracht werden.



PIN	Bezeichnung	PIN	Bezeichnung
1	GND	A	GND
2	+5V	B	CB 1
3	RESET	C	PB 0
4	PA 2 (JOY 0)	D	PB 1
5	PA 3 (JOY 1)	E	PB 2
6	PA 4 (JOY 2)	F	PB 3
7	PA 5 (JOY 4 / LIGHT PEN)	H	PB 4
8	PA 6 (KASS.-SCHALTER)	J	PB 5
9	PA 7	K	PB 6
10	9V AC	L	PB 7
11	9V AC	M	CB 2
12	GND	N	GND.

Bild A.2.3: USER-Port

Die hier aufgefuhrten Port-Leitungen (PA's, PB's und CB's) fuhren zum VIA #1 des VC-20, der im Adreßbereich $9110 bis $911F liegt. Die Bezeichnung JOY 4 steht fur Feuerknopf des Joysticks.

Utility Betriebssystem-Zusatzprogramm

Vektor

es gibt auch andere Begriffe dafür, z.B. Pointer, Zeiger etc. Ein Vektor besteht in der Regel aus 2 Bytes. Dieses Paar ist nichts weiter als eine Adresse, die im Programm den Zweck haben, Sprunge hierdurch indirekt festzulegen oder auf hierdurch adressierte Daten zugreifen zu konnen.

VIA

Versatile **I**nterface **A**dapter (6522); es gibt im VC-20 derer zwei Exemplare:

VIA #1	im Adreßbereich	$9110 - $911F
VIA #2	" "	$9120 - $912F

VIA #2 wird vornehmlich zur Tastaturabfrage verwendet, wogegen VIA #1 fur die eigene Programmierung nahezu frei verfugbar ist.

VIC s. Video-Chip

Video-Chip

Der Video-Chip ist dasjenige Bauteil eines Rechners, welches unabhanging vom Mikroprozessor die Zeichen im Videospeicher in Verbindung mit dem Farbspeicher in für ein Fernsehgerat bzw. Monitor verstandliche Signale umwandelt und sie diesen Geraten zuganglich macht. Beim VC-20 heißt dieser Chip VIC (Video Interface Chip) 6561. Ebenfalls unabhangig von der Prozessorleistung vermag er Tonsignale abzugeben sowie auch Lichtgriffel- und Potentiometersignale aufzunehmen. Weitere Einzelheiten entnehme man diesbezuglich dem Anhang A.5, Tabelle A.5-10.

Video-Speicher

ist derjenige Speicherbereich, in dem die Zeichen des gegenwartigen Bildschirms eingetragen sind. Hierbei sind die Zeichen nicht im ASCII-Code, sondern in einem speziellen Video-Code (s.S.

141,142 im Handbuch) eingetragen, was recht nutzlich ist, weil dadurch samtliche 256 Moglichkeiten fur die Codierung eines Zeichens ausgeschopft werden. Umrechnungs-Routinen zwischen beiden Codes untereinander finden Sie in Kap. 6.11.
Das Pendant zum Video-Speicher ist der Farbspeicher, in dem analog zu den Zeichen im Video-Speicher deren Farben eingetragen sind.
Fur die Versionen GV, sowie ab 8K-V sind das die Bereiche:

GV und 3K-V:	$1E00 - 1FF9	#7680 - 8185
ab 8K-V:	$1000 - 11F9	#4096 - 4601

Es gibt jedoch noch 6 weitere Moglichkeiten (s. Kap. 6.4 "Seitensprung" sowie Tabelle A.5-6).
Eine Berechnungsformel, die den unterschiedlichen Bereichszuweisungen abhangig vom jeweiligen Speicherausbau Rechnung tragt, wird in Kap. 6.3 "Losung des Portabilitats-Problems" angegeben.

V.24-Schnittstelle

V.24 ist die Kurzbezeichnung fur eine CCITT-Empfehlung, die zum Ziel hat, weltweit die Kompatibilitat von Endgeraten fur den Datenaustausch uber Fernsprechnetze zu garantieren. Hinter der Kurzbezeichnung RS 232 verbergen sich die gleichen Schnittstellenregeln. Wegen der weltweiten Unterstutzung dieser Empfehlung wird deren Realisierung in Form von Betriebssystemroutinen (wie auch beim VC-20) nicht nur zur Datenubergabe zu einem Modem (und damit zum Fernsprechnetz), sondern auch zu anderen Peripheriegeraten wie z.B. zu Druckern eingesetzt.
Nach der neueren Norm DIN ISO 7498 fur die modellhafte Beschreibung der Kommunikation zwischen offenen Systemen ist die V.24-Schnittstelle der Schicht 1 (untersten Schicht) zuzuordnen.

Zeichengenerator

Derjenige Speicherbereich, der fur die Darstellung jedes Zeichens die dafur notwendige Information enthalt, namlich deren Anordnung der einzelnen Pixelpunkte. Da ein Zeichen beim VC-20 durch eine 8x8 Pixelmatrix bestimmt ist, sind demzufolge 8 Bytes fur ein Zeichen notig. Denn jedes Bit dieser 8 Bytes kann uber das Vor-

handen- oder Nicht-Vorhanden-Sein eines der 64 Pixels entscheiden. Bit=1 heißt: Pixel ist vorhanden, Bit=0 heißt: Pixel nicht vorhanden. Da beim VC-20 128 Zeichen im Großschrift/Graphik-Modus, 128 im Klein-/Großschrift-Modus, sowie alle diese im Revers-Modus dargestellt werden konnen, sind hierfur (128+128)*2 * 8 = 4096 Bytes erforderlich. Diese liegen beim VC-20 im Bereich $8000-8FFF und sind "von Hause aus" ROM, konnen aber ohne große Muhe im RAM-Bereich adressiert werden, wenn man die ersten vier Bits des VIC-Kontrollregisters CR5 (Adresse $9005 = #36869) verandert (siehe hierzu Tabelle A.5-8).

Zeiger s. Pointer

Zero-Page

ist korrekt definiert als der Speicherbereich $0000-$00FF, d.h. das HIGH-Byte ist $00; jedoch wird beim VC-20 dieser Begriff zumeist fur den Bereich $0000-$00FF und $0200-$03FF benutzt, weil sich in diesem RAM-Bereich die in der Einschaltphase (bzw. nach RESET) festzulegenden und desweiteren dynamisch anderbaren Betriebssystem- bzw. Interpreter-Werte befinden.

A.3 ZERO-PAGE-ADRESSEN

HEX	DEZ	
0B	11	Vektor auf Input-Puffer
0C	12	Flag fur Dimensionierung bzw. Funktionen
0D	13	= $FF fur eingelesenen String = $00 fur eingelesenen numerischen Ausdruck
0E	14	= $80 fur zu bearbeitenden Integer-Wert = $00 fur zu bearbeitenden Gleitkomma-Wert
2B,2C	43,44	Vektor auf Basic-Anfang Default-Werte: GV: $01, 10 = #1, 16 3K-V: $01, 04 = #1, 4 ab 8K-V: $01, 12 = #1, 18 Beispiel: bei einem VC-20 in Grundversion enthält die Speicherzelle #43 den Wert 1 und #44 den Wert #16 unmittelbar nach dem Einschalten oder nach RESET (falls nicht durch eingsteckte Module das Standard-VC-20-Betriebssystem umgangen wird, mehr daruber im Kap. 6.5 "Autostart fur Basic- und Modulprogramme"). Das heißt, die Basic-Anfangsadresse errechnet sich zu: 1 + 16*256 = 4097 (PEEK(43)+PEEK(44)*256)
2D,2E	45,46	Vektor auf Variablen-Anfang Wenn kein Programm geladen ist, hat dieser Vektor den Wert des Vektors $2B,2C zuzuglich 2. In diesem Fall hat z.B. bei einem VC-20 mit 8K-Byte-Zusatzspeicher die Speicherzelle #45 den Wert 3 und #46 den Wert 18.

HEX	DEZ	
2F,30	47,48	Vektor auf Feld-Anfang Wenn keine Felder in einem Basic-Programm benutzt werden, so entspricht dieser Vektor dem Vektor $2D,2E.
31,32	49,50	Vektor auf Feld-Ende Wenn keine Felder in einem Basic-Programm benutzt werden, so entspricht dieser Vektor dem Vektor $2D,2E.
33,34	51,52	Vektor auf untere Grenze des Zeichenspeichers
35,36	53,54	Vektor auf obere Grenze der zuletzt abgespeicherten Zeichenkette
37,38	55,56	Vektor auf totale obere Grenze des Zeichenspeichers = obere Grenze des Basic-Bereichs Default-Werte: GV: $00, 1E = #0, 30 3K-V: $00, 1E = #0, 30 8K-V: $00, 40 = #0, 64 16K-V: $00, 60 = #0, 96 24K-V: $00, 80 = #0,128 Beispielsweise enthalt die Speicherzelle #56 eines um 16K-Byte erweiterten VC-20 den Wert #96.
39,3A	57,58	Momentane Basic-Zeilennummer (L,H) Zeilennummer = PEEK(57)+256*PEEK(58)
3B,3C	59,60	Vorhergehende Basic-Zeilennummer (L,H) (Berechnung siehe daruber)
3F-40	63-64	DATA-Zeilennummer (L,H) Dies ist die Nummer derjenigen DATA-Zeile, aus der Daten entnommen werden (READ-Befehl)
41-42	65-66	Vektor auf das Komma vor dem nachsten Datum der in $3F,40 angegebenen DATA-Zeile

HEX	DEZ	
45,46	69,70	Momentaner Variablenname (Kodierung s. Kap. 2.1 "Speicherorganisation bei einem Basic-Programms")
47,48	71,72	Vektor auf momentane Variable
49,4A	73,74	Vektor auf momentane Laufvariable (FOR-NEXT-Befehl)
61-68	97-104	Fließkomma-Akku #1
69-6E	105-110	Fließkomma-Akku #2
73-8A	115-138	Scanner-Routine; dieses kleine Unterprogramm liegt eigentlich im ROM-Bereich zwischen den Adressen $E387 und $E39E, wird aber wahrend der RESET-Routine in den Bereich $0073-008A kopiert. Abgesehen davon, daß nur dort diese Routine funktioniert, wird wegen Ihrer "Lage" im RAM-Bereich beim "Abgreifen" (Scannen) eine indirekte Adressierung eingespart. Die Register X und Y werden also nicht angetastet, brauchen demnach nicht gerettet werden. Die Scanner-Routine (s. Bild A.3-1) vergroßert den

```
0073 INC $7A
0075 BNE $0079
0077 INC $7B
0079 LDA $HHLL
007C CMP #$3A
007E BCS $008A          (Bild A.3.1)
0080 CMP #$20
0082 BEQ $0073
0084 SEC
0085 SBC #$30
0087 SEC
0088 SBC #$D0
008A RTS
```

Vektor $7A,7B - die Adresse $HHLL also - um 1. greift den zugehorigen Wert in Zeile $0079 ab und wertet ihn in den folgenden Zeilen derart aus, daß diese Routine verlassen wird mit:

Carry-Bit (C) nicht gesetzt:
der Inhalt des abgegriffenen Bytes liegt im Wertebereich $30-39 (ASCII-Code fur Ziffern 0-9)

Carry-Bit gesetzt:
Gegenteil von oben

Zero-Bit (Z) gesetzt:
das abgegriffene Byte hat den Wert $00 oder $3A (Doppelpunkt), was auf ein Befehlsende bei Basic hinweist

Zero-Bit nicht gesetzt:
Gegenteil von oben

Falls Sie allerdings diese Routine fur Ihre Zwecke ausnutzen sollten, mussen Sie peinlich genau auf die Wiederherstellung des Vektors $7A,7B, also die Adresse $HHLL (s. Bild A.3-1), achten, sonst gibt's "SYNTAX ERROR".
Die Scanner-Routine wird ebenso haufig ab Adresse $0079 = #121 mittels JSR $0079 benutzt, um das momentane Zeichen in Adresse $HHLL in den Akku zu holen und die C- und Z-Flags in der beschriebenen Weise zu setzen.

Test-Beispiel:
Die Eingabe: SYS 121:PRINT PEEK(780) (Return) wirft den Dezimalwert 58 aus. Das entspricht dem Wert $3A, der dem Doppelpunkt zwischen den beiden Befehlen entspricht. Da ja nach einem SYS-Befehl der Akku-Inhalt uber die Adresse #780 zur Basic-"Welt" gelangt, heißt das: nachdem ein Basic-Befehl zwecks Bearbeitung interpretiert worden ist, ist der Vektor $7A,7B auf das diesem Befehl unmittelbar folgende Zeichen gerichtet.

HEX	DEZ	
7A-7B	122-123	Vektor auf einzulesendes Byte in der Scanner-Routine $73-8A
A0-A2	160-162	3-Byte-Zahler (VC-Uhr), welcher jede 1/60 s um 1 weitergezahlt wird; A2 enthalt das niederwertigste Byte
B2-B3	178-179	Vektor auf Anfangsadresse des Kassettenpuffers (Default: $033C = #828)
C5	197	Welche Taste gedruckt? (s. Tab. A.5-9) Oft ist es in einem Programm mehr von Bedeutung, die momentan gedruckte Taste zu berucksichtigen. Wird sie wieder losgelassen, soll die Tatsache, daß sie vorher irgendwann mal gedruckt worden ist, keine Rolle mehr spielen. Mit GET und INPUT laßt sich eine solche Steuerung nicht realisieren, weil beide Funktionen diejenigen Zeichen annehmen, welche im Tastaturpuffer zwischengespeichert sind, also auch die in der Vergangenheit eingegebenen. Mittels PEEK(197) laßt sich muhelos bei Zuhilfenahme der Tabelle A.5-9 die momentan gedruckte Taste identifizieren und verwerten. Der Inhalt #64 steht dafur, daß zur Zeit (ausgenommen die CTRL-, CBM- und SHIFT-Tasten) keine Taste gedruckt ist. Die Zustande Gedruckt/Nicht gedruckt der CTRL-, CBM und SHIFT-Tasten konnen uber PEEK(653) (siehe dort) ausgelesen werden.
C6	198	Anzahl der noch abzuarbeitenden Zeichen im Kassettenpuffer (maximal so viele, wie in Adresse $0289 = #649 vom Benutzer festgelegt wird, allerdings nicht mehr als #10, siehe auch $0277-0280)
C7	199	RVS-Flag; Bit 1 gesetzt (=$02), wenn RVS ON

HEX	DEZ	
CB	203	gleiche Bedeutung wie $C5; für den Benutzer haben demnach $C5 und $CB scheinbar die gleiche Bedeutung. Im VC-20-Betriebssystem dient jedoch die eine Zelle als Zwischenpuffer fur die andere, damit es erkennen kann, ob inzwischen eine andere Taste bzw. gar keine gedrückt ist.
CC	204	=1 : Cursor AUS bei GET =0 : Cursor EIN bei GET
CD	205	Cursor-Blinkzähler
CF	207	Flag fur Cursor in RVS-Darstellung
D1-D2	209-210	Vektor auf Anfang derjenigen Zeile im Video-Speicher, in der sich der Cursor befindet
D3	211	Offset (0-87) bzgl. des Vektors $D1,D2; d.h. dieser Wert ist gewissermaßen ein Erganzungsvektor, der auf die Position des Cursors in der durch $D1,D2 angegebenen Zeile zeigt.
D5	213	Anzahl der Zeichen in Zeile, in der sich der Cursor befindet
D6	214	Zeile (0-22), in der sich Cursor befindet
F3-F4	243-244	Vektor auf Anfang derjenigen Zeile im Farbspeicher, in der sich der Cursor befindet. (vgl. $D1,D2); die Routine $EAB2-EABE berechnet $F3,F4 aus $D1,D2
FB-FE	251-254	frei verfugbar
0277-0280	631-640	Tastaturpuffer; das erste abzuarbeitende Zeichen ist in $0277 (siehe auch $C6 und $0289). Oft werden diese Speicherzellen benutzt, um Tastatureingaben zu simulieren, indem man ein-

HEX DEZ

fach ab #631 darin das hineinPOKEd, was man sonst in die Tastatur hineintippen wurde. Allerdings muß man bei diesem Verfahren noch zusätzlich in die Zelle #198 die Anzahl der einzugebenden Zeichen hineinPOKEn.
Beispielsweise soll das Byte #$KK in den Tasturpuffer gelegt werden. Eine M-PGM-Routine konnte so aussehen:

```
        LDA #$KK
        LDX $C6
        CPX $0289
        BCS (Mark)
        STA $0277,X
        INC $C6
Mark:   RTS
```

Wenn das Zeichen nicht abgelegt werden konnte, weil der Tastaturpuffer bereits voll ist, wird obige Routine mit gesetztem Carry-Bit verlassen.

Das entsprechende Basic-Analogon zu obiger Routine konnte sein:

```
1000 C=0:X=PEEK(198):IF X>=PEEK(649) THEN C=1:
     RETURN
1010 POKE 631+X,(DEZ-Wert von $KK):POKE 198,
     PEEK(198)+1:RETURN
```

0281-0282 641-642 Vektor auf Anfang des fur Basic verfugbaren RAM-Bereichs (kann vom Benutzer verändert werden)
Default-Werte:
GV und ab 8K-V: $01, 10 = #1, 16
3K-V: $01, 04 = #1, 4

0283-0284 643-644 Vektor auf Ende des für Basic verfugbaren RAM-Bereichs (kann vom Benuzer verändert werden)

HEX	DEZ	
		Default-Werte: GV und 3K-V: $00, 1E = #0, 30 8K-V: $00, 40 = #0, 64 16K-V: $00, 60 = #0, 96 24K-V: $00, 80 = #0,128
0286	646	Zeichenfarbe (Blau=6; immer 1 weniger, als auf Tastatur angegeben ist)
0288	648	1. Page vom Video-Speicher Default-Werte: #30 in GV, 3K-V #16 in 8K-V, 16K-V, 24K-V
0289	649	Maximale Anzahl der Zeichen, die im Tastaturpuffer stehen durfen (Maximum und Initialwert ist #10); z.B. nach der Eingabe POKE 649,0 wird keine Eingabe mehr von der Tastatur her zugelassen
028A	650	Bit 7 gesetzt (=#128): alle Tasten wiederholen; z.B. POKE 650,222 Bit 6 gesetzt (=#64): keine Tasten wiederholen; z.B. POKE 650,66 Bit 6 und 7 nicht gesetzt: normal z.B. POKE 650,44
028D	653	gedruckte SHIFT, CBM, CTRL-Taste SHIFT: Bit 0 gesetzt (=1) CBM: Bit 1 gesetzt (=2) CTRL: Bit 2 gesetzt (=4) Wenn z.B. CBM- und CTRL-Taste zugleich gedruckt werden, steht in Zelle #653 der Wert 6.
028F-0290	655-656	Vektor auf Tastaturdekodierung (Default: $EBDC)
0291	657	Zeichenumschaltung freigeben/verriegeln frei: Bit 7 nicht gesetzt (Werte #0-127) verriegelt: Bit 7 gesetzt (Werte #128-255)

HEX DEZ

In Verbindung mit dem Verandern des Bit 7 dieser Zero-Page-Adresse lassen sich z.B. die Befehle:

PRINT CHR$(14): Klein-/Großschrift einschalten
PRINT CHR$(142): Großschr./Graphik einschalten

dadurch kombinieren, daß es dem Benutzer eines Programms unmoglich gemacht werden soll, den vom Programm vorgegebenen Schriftmodus durch Drucken von SHIFT- und CBM-Taste zu verandern.

0295 659 Kontrollregister der RS232-Schnittstelle

0296 660 Befehlsregister der RS232-Schnittstelle

029B-029C 665-666 Zeitkonstante der RS232-Schnittstelle

0300-0301 768-769 Vektor zur Fehleranzeige-Routine
(Default: $C43A)
Verandert kann dieser Vektor wie folgt:

```
LDX #$LL
LDY #$HH
STX $0300
STY $0301
RTS
```

In Basic entspräche dies:
POKE 768,(DEZ-Wert von $LL):POKE 769,(DEZ-Wert von $HH)

Das Resultat dieser Veränderung ist, daß in Fehlerfallen (z.B."OUT OF MEMORY" oder "ILLEGAL QUANTITY") nicht zur vorgesehenen Adresse $C43A gesprungen wird, sondern zu der hier festgelegten Adresse $HHLL.

HEX	DEZ	
0302- 0303	770- 771	Vektor auf Einsprung in die Basic-Warteschleife (Default: $C483); der Einsprung wird auch als "Warmstart" bezeichnet.
0304- 0305	772- 773	Vektor zur Routine für die Umwandlung einer Zeile in Basic-PGM-Code (Default: $C57C)
0306- 0307	774- 775	Vektor zur Routine fur die Ausgabe der den Basic-Token entsprechenden Befehlsworte (Default: $C71A)
0308- 0309	776- 777	Vektor zum Einsprung in die Basic-Token-Interpretier-Routine (Default: $C7E4); zur Anderung dieses Vektors siehe $0300
030A- 030B	778- 779	Vektor zur Routine fur die Auswertung eines Terms (Default: $CE86)
030C	780	im SYS-Befehl übergebenes A-Register
030D	781	im SYS-Befehl übergebenes X-Register
030E	782	im SYS-Befehl übergebenes Y-Register
030F	783	im SYS-Befehl übergebenes PSW-Register
0314- 0315	788- 789	Vektor zur IRQ-Routine bei nicht gesetztem BRK-Flag (Default: $EABF); man kann diesen Vektor zwar auch verändern, jedoch geht das nicht einfach durch POKEn in diese beiden Speicherzellen. Was hier noch berucksichtigt werden muß, ist, daß die Umanderung von beiden Werten gar nicht so schnell erfolgen kann, wie die Interrupt-Routine nach 1/60 s erneut aufgerufen wird. D.h. nach Veranderung eines der beiden Werte springt die Interrupt-Routine zu einer undefinierten Adresse, was in den meisten Fallen zum totalen Systemabsturz fuhrt. Das I-Flag (Interrupt-Disable-Flag) muß hierbei also gesetzt werden. Dieses Flag wird in Basic

uber einen einfachen Trick beeinflußt (in Maschinensprache gabe es hier ohnehin keine Probleme, weil es den Befehl SEI gibt): uber die Adressen #780-783 erhalten die Register A,X,Y, PSW mittels Befehl SYS die vom Benutzer gewunschten Werte. Bevor also in die Adressen #788,789 hineingePOKEd wird, setzt man in die Speicherzelle #783 den Wert 4 (was ja bedeutet: Bit 2 setzen) und springt mit SYS zu einem beliebigen Rucksprungbefehl (in Maschinensprache ist das der Befehl RTS mit dem Code #96), z.B. SYS 60044. Prufen Sie's nach! uberzeugen Sie sich, daß in #60044 wirklich der Wert #96 steht. Nachdem auf diese Weise das I-Flag, gewissermaßen durch die Hintertur gesetzt wurde, kann der Vektor #788,789 ohne Bedenken "verbogen" werden.
Zum Beispiel soll dieser Vektor auf $2000=#8192 zeigen (LOW=0, HIGH=32; 0+32*256=8192), von wo ab irgendein M-PGM beginnt. Er wird folgendermaßen verbogen:

```
POKE 783,4:SYS 60044:POKE 788,0:POKE 789,32 (Return)
```

Zu beachten ware hierbei lediglich, daß diese 4 Befehle alle in einer Befehlszeile sind (jedenfalls im Direkt-Modus).

Im M-PGM sahe die Vektorveranderung folgendermaßen aus:

```
LDX #$LL
LDY #$HH
SEI
STX $0314
STY $0315
RTS
```

Nach dieser Befehlsfolge zeigt der IRQ-Vektor auf die Adresse $HHLL.

HEX	DEZ	
0316- 0317	790- 791	Vektor zur IRQ-Routine bei gesetztem BRK-Flag (Default: $FED2)
0318- 0319	792- 793	Vektor zur NMI-Routine (Default: $FEAD)
0326- 0327	806- 807	Vektor zur Output-Routine (Default: $F27A)
0328- 0329	808- 809	Vektor zur Routine für die Abfrage der STOP-Taste (Default: $F770). Mittels "Verbiegen" dieses Vektors, z.B. durch POKE 808,100, bleibt das Drucken der Tasten STOP sowie RESTORE wirkungslos. Per POKE 808,112 wird der Anafangszustand wiederhergestellt.
032A- 032B	810- 811	Vektor auf Input-Routine (Default: $F1F5)
032E- 032F	814- 815	frei verfügbar
0334- 033B	820- 827	frei verfügbar

A.4 BETRIEBSSYSTEM-ROUTINEN

In diesem Anhang-Teil werden hauptsachlich diejenigen Betriebssystem-Routinen eingehender beprochen, welche auch in den in diesem Buch veröffentlichten Programmen benutzt werden. Die folgenden Auflistung erhebt deswegen keinen Vollstandigkeitsanspruch. Eine vielleicht langere, jedoch dafur nicht mit so zahlreichen Erklärungen versehenen Liste von Betriebssystem-Routinen ist bereits an anderer Stelle veröffentlicht.

Über der jeweiligen Erläuterung stehen die Adressen auf der linken Seite in HEX- und auf der rechten Seite in DEZ-Schreibweise.

C09E - C19C **49310 - 49564**

Siehe Tabelle A.5.3 im Anhang A.5. In diesem Bereich sind samtliche Basic-Befehle und -Funktionsworte abgespeichert.

C38A **50058**

Stapelsuchroutine für die Befehle FOR-NEXT und GOSUB

C3FB **50171**

Die Routine ab $C3FB pruft, ob noch Platz im Stack ist.

C435 **50229**

Ausgabe "OUT OF MEMORY" und "READY", Rückkehr zum Direkt-Modus, siehe zusatzlich $C437 und $C43A.

C437 **50231**

Sprung zur Fehlermeldungs-Routine. Im Betriebssystem steht hier der Befehl JMP ($0300). Der Einsprung in diese Routine wird demnach durch den Vektor in $0300,$0301 (L,H) vorgegeben. Nach Ab-

lauf der RESET-Routine steht hier der Default-Wert $C43A, er kann aber auch von Ihnen verandert werden:

```
im M-PGM:  LDX #$LL        ($HHLL sei die von Ihnen
           LDY #$HH         gewunschte Einsprungadresse)
           STX $0300
           STY $0301
           RTS

im B-PGM:  POKE 768,L (DEZ-Wert von $LL):
           POKE 769,H (DEZ-Wert von $HH)
```

C43A **50234**

Ausgabe einer Fehlermeldung und "READY" und Ruckkehr zum Direkt-Modus. Die Fehlermeldung wird durch den im X-Register (Fehlernummer) ubergebenen Wert spezifiziert. Eine Gegenuberstellung von Fehlernummern und -meldungen finden Sie in Anhang A.5, Tabelle A.5.4. Die Benutzung dieser Routine in eigenen Programmen geschieht mittels zwei Befehlen:

```
im M-PGM:  LDX #$NN       ($NN ist die Fehlernummer)
           JMP $C43A

im B-PGM:  POKE 781,N:SYS 50234   (N ist die DEZ-Fehlernummer)
```

C474 **50292**

Einsprung in den READY-Modus (PGM-Austritt)

C480 **50304**

Vektorisierter Sprung (Vektor in $0302,$0303) auf die Routine "Eingabe-Warteschleife". Die vom Betriebssystem vorgegebene Einsprungadresse ist $C483, kann aber entsprechend der unter $C437 gemachten Angaben verandert werden.

C483 **50307**

Einsprung in die Basic-Eingabe-Warteschleife; eigentlich geschieht dies indirekt schon mittels Befehl in $C480-C482:

```
JMP ($0302)
```

wenn der Vektor $0302,0303 auf die Adresse $C483 zeigt.

C49C **50332**

Einfügen und Loschen einer Programmzeile

C613 **50707**

Berechnung der Startadresse einer Basic-Programmzeile, deren Zeilennummer $HHLL (HEX-Darstellung !) in den Speicherzellen $14,15 (L,H) steht. Die Anfangsadresse steht nach Ruckkehr aus dieser Routine in den Zellen $5F,60 (L,H). Falls die Basic-Zeile nicht gefunden wurde, wird die Routine mit geloschtem Carry-Flag verlassen. Die Routine konnte in folgendem M-UP benutzt werden:

```
                    LDX #$LL
                    LDY #$HH
-----------------------------
UP-Einsprung:       STX $14
                    STY $15
                    JSR $C613
                    BCC Mark1
                    LDX $5F
                    LDY $60
                    RTS
          Mark1:    LDX #$FF
                    LDY #$FF
                    RTS
```

Die Basic-Zeilennummer wird beim Einsprung in das UP in den X,Y-Registern (L,H) übergeben. Nach Ruckkehr aus dem UP enthalten die X-,Y-Register die gesuchte Startadresse der Basic-Zeile oder aber beide den Wert $FF, falls die Zeilennummer im B-PGM nicht gefunden worden ist.

C68E **50830**

Der Scanner-Vektor $7A,7B erhält den um 1 verminderten Wert des Vektors $2B,2C.

C69C **50844**

Bearbeitung des Befehls Basic-Befehls LIST. Bei evtl. Benutzung dieser Routine in M-PGM'en muss dem JSR $C69C allerdings unbedingt der Befehl JSR $0079 vorangehen.

C7AE **51118**

Anfang der Interpreter-Routine

C7E4 **51172**

Interpretieren des Basic-Token

C82C **51244**

Stop-Taste abfragen und Basic-Stop veranlassen, falls gedruckt. Im Falle einer STOP-Anweisung im B-PGM wird die Stop-Tastenabfrage (JSR $FFE1) umgangen und in $C82F eingesprungen.

C8A3 **51363**

GOTO-Routine; in $14,15 muß die Ziel-Zeilennummer (L,H) stehen.

C906 **51462**

Bestimmung des Endes eines Befehls innerhalb einer Zeile. In Y steht der Offset bzgl. des Zeilenanfangs zum Doppelpunkt (Endekennzeichen).

C96B **51563**

Zeilennummer einlesen, ins LOW/HIGH-Format umwandeln und in $14. 15 abspeichern. Vektor $7A,7B muß auf die 1. Ziffer des ASCII-Ziffernstring zeigen.

C9A5 **51621**

Variablenzuweisungsbefehl; z.B.: LET A=5 oder A=5

CB1E **51998**

String-Ausgabe-Routine; ein String, der ab Adresse $HHLL gespeichert ist und eine 0 als Endekennzeichen aufweist, wird ausgegeben (Bildschirm oder Drucker).

```
im M-PGM:     LDA #$LL
              LDY #$HH
              JSR $CB1E
im Basic:     POKE 780,(DEZ-Wert von $LL):POKE 782,(DEZ-Wert von
              $HH):SYS 51998
```

Zum Beispiel kann per:

POKE 780,65:POKE 782,3:SYS 51998 (Return)

der Kassettenpuffer bis zur ersten 0 ausgelesen werden. Dies kann von Nutzen sein, will man den gesamten Programmnamen erfahren, unter dem ein Programm abgespeichert worden ist.

Folgende Meldungen lassen sich nach obigen Verfahren ausgeben:

$-Adresse	#-LOW,	#-HIGH	Meldung
CCFC	252	204	? EXTRA IGNORED
CD0C	12	205	? REDO FROM START
C364	100	195	OK
C369	105	195	ERROR
C36B	107	195	ERROR (ohne WR und Space am Anfang)
C371	113	195	IN
C376	118	195	READY.
C381	129	195	BREAK
E429	41	228	BYTES FREE

CB3F **52031**

Ausgabe eines Leerzeichens

CB42 **52034**

Cursor wird um eine Stelle nach rechts bewegt.

CB45 **52037**

Ausgabe eines Fragezeichens

CB47 **52039**

Ausgabe des im Akku stehenden Zeichens

CD8A **52618**

Numerischen Term auswerten; der Vektor $7A,7B muß auf das 1. Zeichen des Terms zeigen.

CD8D **52621**

prüft eingelesenen Term auf numerisch; wenn er nicht numerisch ist, folgt die Ausgabe der Fehlermeldung "TYPE MISMATCH".

CD8F **52623**

prüft eingelesenen Term auf String; wenn er nicht vom Typ String ist, folgt die Ausgabe der Fehlermeldung "TYPE MISMATCH".

CD9E **52638**

beliebigen Term auswerten; der Vektor $7A,7B muß auf das 1. Zeichen des Term zeigen. Ausgang aus der Routine: in $0D steht $FF,

wenn der Term ein String ist; in $45,46 steht der Name und in $47,48 die Adresse der Variablen, welche als letzte im eingelesenen Term vorkommt.

CEF7 **52983**

momentanes Zeichen, auf das der Vektor $7A,7B zeigt, auf "Klammer auf" prüfen und nächstes Zeichen holen. Falls nicht "Klammer auf", folgt "SYNTAX ERROR".

CEFA **52986**

analog zu $CEF7 sinngemaß fur "Klammer zu"

CEFD **52989**

prüfen auf Komma; sinngemäß wie bei $CEF7

CEFF **52991**

vergleicht das im Akku stehende Zeichen mit dem Zeichen, auf das der Vektor $7A,7B zeigt. Falls es nicht identisch ist, folgt die Meldung "SYNTAX ERROR".

CF08 **53000**

Ausgabe der Fehlermeldung "SYNTAX ERROR" und Sprung in den READY-Modus (Austritt aus dem Programm). Kann in Basic per SYS 53000 benutzt werden.

CF4B **53067**

Umrechnung eines 3-Byte-Zahler-Wertes:

$62	enthält 0
$63	höchstwertiges Byte
$64	mittelwertiges Byte
$65	niederwertiges Byte

in die Uhrzeit und dessen ASCII-String erzeugen; wird in Verbindung mit der Routine ab $D6A6 benutzt, die den String-Deskriptor nach Register A,X,Y (Länge,L,H) legt.

CF87 **53127**

Der Wert eines 3-Byte-Zählers wird in den Fließkomma-Akkumulator (Bereich $61-68) geladen:

#$00	-->	$62	
Wert von Y	-->	$63	hochstwertiges Byte
Wert von X	-->	$64	mittelwertiges Byte
Wert von A	-->	$65	niederwertiges Byte

D113 **53523**

prüft Zeichen im Akku auf Buchstabe; das Carry-Flag ist gesetzt, wenn es ein Buchstabe ist.

D245 **53829**

Ausgabe der Fehlermeldung "BAD SUBSCRIPT ERROR" und Sprung in den READY-Modus (PGM-Austritt).

D248 **53832**

sinngemäß wie D245 für die Fehlermeldung "ILLEGAL QUANTITY"

D6A6 **54950**

siehe CF4B; String-Deskriptor wird in die Register A,X,Y (Länge,L,H) gebracht.

D7F7 **55287**

Umwandlung einer Integer-Zahl von 0 bis 65535 ins LOW/HIGH-Format. Die zwei berechneten Bytes liegen in den Zellen $14,15 (L,H) und in den Registern Y,A.

Diese Routine wird in Verbindung mit $CD8A verwendet, so daß mit der Kombination:

```
JSR $CD8A
JSR $D7F7
```

eine in Basic (PGM oder Direkt-Modus) eingegebene Integer-Zahl bequem in ein Maschinenprogramm übergeben werden kann.

DDCD **56781**

Ausgabe einer Integer-Zahl zwischen 0 und 65535, wobei das 2-Byte-Pendant (L,H) in den Registern X,A vorgegeben wird.
In M-PGM wird diese Routine wie folgt verwendet:

```
LDX #$LL
LDY #$HH
JSR $DDCD
```

($HHLL sei die auszugebene Zahl in HEX).

Das entsprechende Basic-Analogon:
POKE 781,(DEZ-Wert von $LL):POKE 782,(DEZ-Wert von $HH):SYS 56781
läßt sich natürlich einfacher durchführen mit:

```
PRINT (DEZ-Wert von $LL)+256*(DEZ-Wert von $HH)
```

DDDD **56797**

Eine im Fließkomma-Akku (Bereich $61-68) stehende Zahl wird in ein ASCII-String umgewandelt und das Ergebnis nach $0100-... (abhanging von der Länge) gebracht.

E251 **57937**

liest einen String oder eine Stringvariable ein und speichert den Deskriptor (Länge, Anfangsadresse L/H) des Strings nach $B7, $BB, $BC und A,X,Y. Hierbei muß der Vektor $7A,7B auf ein Komma vor dem String oder der Stringvariablen zeigen.

Will man in Basic den Deskriptor eines Strings ermitteln, so gibt man ein:
SYS 57937,(String od. Stringvariable):PRINT PEEK(780);PEEK(781); PEEK(782)
Wenn der String z.B. "ABCDE" ist, muß als erster Wert 5 erscheinen, weil die Zelle #780 als Übergabespeicherzelle den Akku-Inhalt, demnach also die Länge des Strings enthalten muß. Von der Richtigkeit der zwei restlichen Zahlen konnen Sie sich einfach so überzeugen:

```
PRINT PEEK((mittlere Zahl)+256*(letzte Zahl)+0);
      PEEK((mittlere Zahl)+256*(letzte Zahl)+1);
      PEEK((mittlere Zahl)+256*(letzte Zahl)+2);
      PEEK((mittlere Zahl)+256*(letzte Zahl)+3);
      PEEK((mittlere Zahl)+256*(letzte Zahl)+4)
```

die Ergebnisse 65,66,67,68,69, nämlich die ASCII-Codes fur: A, B, C, D, E erhalten.

E254 **57940**

siehe E251; der Unterschied dazu ist, daß die Komma-Prüfung entfällt. Der Vektor $7A,7B muß auf das 1. Zeichen des Strings oder der Stringvariablen zeigen.

E387 - E39E **58247 - 58270**

Scanner-Routine; diese funktioniert aber nicht in diesem Adreßbereich, sondern muß in einem beliebigen RAM-Bereich (relocatable) stehen. Beim VC-20 wird diese Routine während der RESET-Routine in den Bereich $73-8A gebracht (siehe weitere Informationen im Anhang A.3 unter $73-8A).

E45B **58459**

Die Basic-Interpreter-Vektoren $0300-030B werden auf die Ausgangswerte (Default-Werte) zurückgesetzt. Per Basic mittels SYS 58459 können z.B. zusätzliche Interpreter-Routinen oder eigene Fehlermeldungsroutinen deaktiviert werden.

E50C **58636**

Cursor setzen; Register X,Y enthalten die vorgegebene Reihen- (0...22) und Spaltennummer (0...21). Per Basic ist die Routine gemäß folgendem Beispiel verwendbar:

```
POKE 781,(Zeile):POKE 782(Spalte):SYS 58636:PRINT "CURSOR"
```

E513 **58643**

Cursor-Position abrufen; Register X,Y enthalten die Reihen- (0...22) und Spaltennummer (0...21). Nach SYS 58643 steht in den Speicherzellen #781, #782 die beim Aufruf gultige Cursor-Position (Zeilen-/Spaltennummer, siehe E50C).

E55F **58719**

entspricht der Funktion CLR HOME

E581 **58753**

entspricht der Funktion CRSR HOME

E5CF **58831**

das nachste Zeichen aus dem Tastaturpuffer holen und in den Akku laden

E975 **59765**

SCROLLING-Funktion. Mit SYS 59765 wird der gesamte Bildschirminhalt um eine Zeile nach oben geSCROLLed. Die oberste Zeile verschwindet und eine leere Zeile ruckt als letzte Zeile nach.

EA8D **60045**

Die im X-Register vorgegebene Zeile (0...22) wird auf dem Bildschirm geloscht. Anwendung:

```
im M-PGM:        LDX #$ZZ
                 JSR $EA8D

im Basic-PGM:    POKE 781,DEZ-Wert von ($ZZ):SYS 60045
```

z.B. wird mit POKE 781,1:SYS 60045 die 2. Zeile gelöscht.

EA8F **60047**

Die im X-Register vorgebene Zeile (0...22) wird bis zur im Y-Register vorgegebenen Spalte (0...21) gelöscht. Anwendung:

```
im M-PGM:        LDX #$ZZ
                 LDY #$SS
                 JSR $EA8F

im Basic-PGM:    POKE 781, (DEZ-Wert zu $ZZ):POKE 782, (DEZ-WERT
                 zu $SS):SYS 60047
```

z.B. werden mit POKE 781,5:POKE 782,7:SYS 60047 die ersten 8 Zeichen der 6. Zeile gelöscht.

EAB2 - EABE **60082 - 60094**

Berechnung des Farb-Speicher-Vektors $F3,F4 aufgrund der Vorgabe des Video-Speicher-Vektors $D1,D2, der auf die Anfangsadresse einer Bildschirmzeile gerichtet ist.

EABF **60095**

Anfang der Interrupt-Routine (s. Anhang A.2), die bei nicht gesetztem BRK-Flag und einer IRQ-Interrupt-Anforderung angesprungen wird, vorausgesetzt, daß der Vektor $0314,0315 auf $EABF zeigt (siehe weitere Angaben hierzu in Anhang A.3, Adressen $0314, 0315).

EB1E **60190**

Einsprung in die Tastatur-Abfrage

EBDC **60380**

Einsprung in die Tastatur-Entschlusselung

F1E2 **61922**

s. Anhang A.5, Tabelle A.5-5

F1F5 **61941**

ist das M-PGM-Pendant zum Basic-Befehl GET. Im Akku-Register steht $00, wenn nichts oder eine $00 im Tastaturpuffer stand. Der Aufruf JSR $F1F5 entspricht JSR $FFE4, wenn Vektor $032A,032B auf $F1F5 zeigt.

F27A **62074**

Ausgabe des im Akku stehenden Zeichens auf das momentan adressierte Ausgabegerat. JSR $F27A entspricht JSR $FFD2, wenn Vektor $0326,0327 auf $F27A zeigt.

F2BA **62138**

Ausgabe eines Zeichens auf Gerat 2 (RS232-Schnittstelle). Das im Akku stehende Zeichen wird in den Ausgabepuffer der RS232-Schnittstelle geschrieben. Der Ausgabepuffer wird mit der OPEN-Anweisung eingerichtet und seine Anfangsadresse in die ZeroPage-Adresse $F9 und FA abgelegt. Sobald der Ausgabepuffer voll ist (max. 256 Zeichen), wird so lange in der Routine gewartet, bis ein Zeichen ausgegeben wurde und damit Platz fur das neue Zeichen geschafft wurde.
Bei kleinen Baud-Raten und großen Mengen auszugebender Daten kann es deshalb zu einem verlangsamten Abarbeiten des Haupt-PGM's fuhren.

Die CLOSE-Anweisung hat zur Folge, daß Daten die noch im Ausgabepuffer stehen und auf ihre Aussendung warten, verloren sind. Eine Prufung von Bit-Nr. 6 im Unterbrechungs-Freigabe-Register der VIA #1 ist deshalb unbedingt vor der CLOSE-Anweisung durchzuführen. Bit 6 ist 0, wenn der Ausgabepuffer leer ist.

F604 **62980**

Programm ohne Header per SYS 62980 laden. Bei Aufruf dieser Routine wird vorausgesetzt, daß die Header-Information (evtl. von Ihnen gezielt verandert) bereits in den Zellen $033C-0340 = #828-832 (s. Kap. 3.2 "Tape-Header") vorliegt. Zum Laden des Headers siehe $F7AF.

F659 **63065**

Ausgabe eines Strings, dessen Deskriptor in $B7, $BB, $BC steht.

F7AF **63407**

Einlesen des Tape-Headers per SYS 63407. Nachdem der Header gelesen worden ist, kann die in den Zellen $033C-0340 gespeicherte Information nach Ihren Wunschen verandert werden, bevor das eigentliche Programm mittels SYS 62980 (s. $F604) geladen wird.

F894 **63636**

gibt "PRESS PLAY ON TAPE" aus und wartet, bis die PLAY-Taste der Datassette gedruckt wird.

F8B7 **63671**

gibt "PRESS RECORD & PLAY ON TAPE" aus und wartet, bis PLAY-Taste der Datassette gedruckt wird.

FD22 **64802**

Einsprung in die RESET-Routine

FD2F **64815**

Teil-RESET (siehe FD22); der automatische Sprung zur in Vektor $A000,A001 angegebenen Adresse wird mittels SYS 64815 verhindert.

FD52 **64850**

Vektoren $0314-0333 auf Ausgangswerte (Default-Werte) setzen

FDCA **64970**

In der Befehlsfolge:
```
LDX #$00
JSR $FDCA
```
wird der RAM-Ende-Vektor $0283,0284 und der Video-Speicher-Anfang auf $1E00 gesetzt.

FDD2 **64978**

In der Befehlsfolge:
```
LDX #$LL
LDY #$HH
JSR $FDD2
```
wird der RAM-Anfang-Vektor $0281,0282 auf $1200, der Video-Speicher-Anfang auf $1000 und der RAM-Ende-Vektor auf $HHLL gesetzt.

FE49 **65097**

Die Inhalte in Akku, X, Y werden in die Speicherplatze $B7, $BB, $BC (Filenamen-Länge und Anfangsadresse) kopiert.

FE50 **65104**

Die Inhalte in Akku, X, Y werden in die Speicherplatze $B8, $BA, $B9 (Filenummer, Primär- und Sekundaradresse) kopiert.

FE8A **65162**

In der Befehlsfolge:
```
LDX #$LL
LDY #$HH
JSR $FE8A
```
wird der RAM-Anfang-Vektor $0281,0282 auf $HHLL gesetzt.

FE91 **65169**

JSR $FE91 pruft eine durch Vektor $C1,C2 adressierte Speicherzelle auf RAM. Das Carry-Flag wird gesetzt, wenn die Speicherzelle RAM ist.

FEA9 **65193**

Einsprung in die NMI-Routine

FED2 **65234**

Anfang der IRQ-Routine bei gesetztem BRK-Flag

FF72 **65394**

Einsprung in die IRQ-Routine

FFBA **65466**

siehe FE50

FFBD **65469**

siehe FE49

FFD2 **65490**

Sprung zur in Vektor $0326,0327 angegebenen Adresse; s. F27A

FFE4 **65508**

Sprung zur in Vektor $032A,032B angegebenen Adresse; s. F1F5

FFE7 **65511**

Bei Ansprung dieser Adresse (SYS 65511) werden alle momentan geöffneten Files geschlossen. Wenn Ihnen also das für den VC-20-Drucker notwendige PRINT#4:CLOSE4 (hierbei wird 4 als Filenummer unterstellt) nicht schnell genug erscheint, kann ebenso SYS 65511 benutzen.

FFF0 **65520**

siehe E50C

A.5 TABELLEN

Tabellen A.5.1 und A.5.2 können recht nützliche Hilfen für Maschinensprache-Programmierer sein, natürlich auch für diejenigen, die darin einsteigen wollen. Zwar wird man solche Tabellen in mehr oder weniger anderer Form auch in anderen speziell für Maschinensprache kompetenteren Büchern finden, aber sie sollten dennoch unserer Meinung nach nicht in einem solchen Buch fehlen, worin sich ja doch eine Menge M-PGMe befinden.

Da es seit einigen Monaten die CMOS-Version der 6502-CPU mit dem Namen G65SC02 gibt, die vollständig aufwärtskompatibel zum 6502 ist (d.h. der 6502 ist aus der VC-20-Grundplatine herauszuziehen und durch den G65SC02 zu ersetzen; das VC-20-Betriebssystem wird danach gleichermaßen funktionieren wie vorher auch), darüberhinaus sogar 27 weitere Befehlscodes aufweist, hielten wir es für richtig, auch gleich diese mit in die Tabellen aufzunehmen. Die hinzugekommenen Befehle sind durch ein + gekennzeichnet. Hinweise zu den zusätzlichen Befehlen befinden sich im Anschluß an die Tabellen A.5.1 und A.5.2.

Des öfteren möchte man gerne wissen, welche Flags bei einem bestimmten Befehl beeinflußt werden oder wieviel Bytes für die Kodierung eines Befehls benötigt werden. Schnell liefert hierbei Tabelle A.5.1 Auskunft, weil darin sämtliche 56 6502-Befehlsarten - es sind insgesamt 151 Befehle, berücksichtigt man noch zusätzlich die unterschiedlichen Adressierungsarten - in alphabetischer Reihenfolge aufgelistet sind. Der Übersichtlichkeit halber ist ein Befehlswort, z.B. LDA, nur einmal aufgeführt und in eingerückter Darstellung die dazugehörigen Adressierungsarten. Analog dazu wurden die beeinflußten Flags auch nur einmal aufgelistet, gelten aber für alle Adressierungsmöglichkeiten. Die Bedeutung der Abkürzungen sind Anhang A.1 zu entnehmen.

Tabelle A.5.2 liefert eine Gegenüberstellung der DEZ- und HEX-Zahlen in aufsteigender Reihenfolge mit den zugehörigen 6502-Befehlen. Da hierbei die Lücken - es gibt nun mal nicht 256 Befehle - nicht ausgelassen wurden, kann die Tabelle A.5.2 ebenso gut als Vergleichstabelle zwischen HEX- und DEZ-Zahlen ihre Dienste tun.

Tabelle A.5.1: **6502-Befehle in alphabetischer Reihenfolge**

Befehl-Mnemonic	HEX-Zahl $	DEZ-Zahl #	Anzahl der Befehlsbytes	Beeinflußte Flags
ADC #$KK	69	105	2	N,V,Z,C
$HHLL	6D	109	3	
$ZZ	65	101	2	
($ZZ,X)	61	97	2	
($ZZ),Y	71	113	2	
$ZZ,X	75	117	2	
$HHLL,X	7D	125	3	
$HHLL,Y	79	121	3	
+ ($ZZ)	72	114	2	
AND #$KK	29	41	2	N,Z
$HHLL	2D	45	3	
$ZZ	25	37	2	
($ZZ,X)	21	33	2	
($ZZ),Y	31	49	2	
$ZZ,X	35	53	2	
$HHLL,X	3D	61	3	
$HHLL,Y	39	57	3	
+ ($ZZ)	32	50	2	
ASL A	0A	10	1	N,Z,C
$HHLL	0E	14	3	
$ZZ	06	6	2	
$ZZ,X	16	22	2	
$HHLL,X	1E	30	3	
BCC	90	144	2	---
BCS	B0	176	2	---
BEQ	F0	240	2	---

Befehl-Mnemonic	HEX-Zahl $	DEZ-Zahl #	Anzahl der Befehlsbytes	Beeinflußte Flags
BIT $HHLL	2C	44	3	N=B7,V=B6,Z
$ZZ	24	36	2	
+ #$KK	89	137	2	
+ $ZZ,X	34	52	2	
+ $HHLL,X	3C	60	3	
BMI	30	48	2	---
BNE	D0	208	2	---
BPL	10	16	2	---
+ BRA	80	128	2	---
BRK	00	0	1	B=1,I=1
BVC	50	80	2	---
BVS	70	112	2	---
CLC	18	24	1	C=0
CLD	D8	216	1	D=0
CLI	58	88	1	I=0
CLV	B8	184	1	V=0
CMP #$KK	C9	201	2	N,Z,C
$HHLL	CD	205	3	
$ZZ	C5	197	2	
($ZZ,X)	C1	193	2	
($ZZ),Y	D1	209	2	
$ZZ,X	D5	213	2	
$HHLL,X	DD	221	3	
$HHLL,Y	D9	217	3	
+ ($ZZ)	D2	210	2	

Befehl-Mnemonic	HEX-Zahl $	DEZ-Zahl #	Anzahl der Befehlsbytes	Beeinflußte Flags
CPX #$KK	E0	224	2	N,Z,C
$HHLL	EC	236	3	
$ZZ	E4	228	2	
CPY #$KK	C0	192	2	N,Z,C
$HHLL	CC	204	3	
$ZZ	C4	196	2	
DEC $HHLL	CE	206	3	N,Z
$ZZ	C6	198	2	
$ZZ,X	D6	214	2	
$HHLL,X	DE	222	3	
+ A	3A	58	1	
DEX	CA	202	1	N,Z
DEY	88	136	1	N,Z
EOR #$KK	49	73	2	N,Z
$HHLL	4D	77	3	
$ZZ	45	69	2	
($ZZ,X)	41	65	2	
($ZZ),Y	51	81	2	
$ZZ,X	55	85	2	
$HHLL,X	5D	93	3	
$HHLL,Y	59	89	3	
+ ($ZZ)	52	82	2	
INC $HHLL	EE	238	3	N,Z
$ZZ	E6	230	2	
$ZZ,X	F6	246	2	
$HHLL,X	FE	254	3	
+ A	1A	26	1	
INX	E8	232	1	N,Z
INY	C8	200	1	N,Z

Befehl-Mnemonic	HEX-Zahl $	DEZ-Zahl #	Anzahl der Befehlsbytes	Beeinflußte Flags
JMP $HHLL	4C	76	3	---
($HHLL)	6C	108	3	---
+ ($HHLL,X)	7C	124	3	---
JSR $HHLL	20	32	3	---
LDA #$KK	A9	169	2	N,Z
$HHLL	AD	173	3	
$ZZ	A5	165	2	
($ZZ,X)	A1	161	2	
($ZZ),Y	B1	177	2	
$ZZ,X	B5	181	2	
$HHLL,X	BD	189	3	
$HHLL,Y	B9	185	3	
+ ($ZZ)	B2	178	2	
LDX #$KK	A2	162	2	N,Z
$HHLL	AE	174	3	
$ZZ	A6	166	2	
$ZZ,Y	B6	182	2	
$HHLL,Y	BE	190	3	
LDY #$KK	A0	160	2	N,Z
$HHLL	AC	172	3	
$ZZ	A4	164	2	
$ZZ,X	B4	180	2	
$HHLL,X	BC	188	3	
LSR A	4A	74	1	N=0,Z,C
$HHLL	4E	78	3	
$ZZ	46	70	2	
$ZZ,X	56	86	2	
$HHLL,X	5E	94	3	
NOP	EA	234	1	---

	Befehl-Mnemonic	HEX-Zahl $	DEZ-Zahl #	Anzahl der Befehlsbytes	Beeinflußte Flags
	ORA #$KK	09	9	2	N,Z
	$HHLL	0D	13	3	
	$ZZ	05	5	2	
	($ZZ,X)	01	1	2	
	($ZZ),Y	11	17	2	
	$ZZ,X	15	21	2	
	$HHLL,X	1D	29	3	
	$HHLL,Y	19	25	3	
+	($ZZ)	12	18	2	
	PHA	48	72	1	---
	PHP	08	8	1	---
+	PHX	DA	218	1	---
+	PHY	5A	90	1	---
	PLA	68	104	1	N,Z
	PLP	28	40	1	alle Flags
+	PLX	FA	250	1	N,Z
+	PLY	7A	122	1	N,Z
	ROL A	2A	42	1	N,Z,C
	$HHLL	2E	46	3	
	$ZZ	26	38	2	
	$ZZ,X	36	54	2	
	$HHLL,X	3E	62	3	
	ROR A	6A	106	1	N,Z,C
	$HHLL	6E	110	3	
	$ZZ	66	102	2	
	$ZZ,X	76	118	2	
	$HHLL,X	7E	126	3	

Befehl-Mnemonic	HEX-Zahl $	DEZ-Zahl #	Anzahl der Befehlsbytes	Beeinflußte Flags
RTI	40	64	1	alle Flags
RTS	60	96	1	---
SBC #$KK	E9	233	2	N,Z,C,V
$HHLL	ED	237	3	
$ZZ	E5	229	2	
($ZZ,X)	E1	225	2	
($ZZ),Y	F1	241	2	
$ZZ,X	F5	245	2	
$HHLL,X	FD	253	3	
$HHLL,Y	F9	249	3	
+ ($ZZ)	F2	242	2	
SEC	38	56	1	C=1
SED	F8	248	1	D=1
SEI	78	120	1	I=1
STA $HHLL	8D	141	3	---
$ZZ	85	133	2	
($ZZ,X)	81	129	2	
($ZZ),Y	91	145	2	
$ZZ,X	95	149	2	
$HHLL,X	9D	157	3	
$HHLL,Y	99	153	3	
+ ($ZZ)	92	146	2	
STX $HHLL	8E	142	3	---
$ZZ	86	134	2	
$ZZ,Y	96	150	2	
STY $HHLL	8C	140	3	---
$ZZ	84	132	2	
$ZZ,X	94	148	2	

Befehl-Mnemonic	HEX-Zahl $	DEZ-Zahl #	Anzahl der Befehlsbytes	Beeinflußte Flags
+ STZ $HHLL	9C	156	3	---
$ZZ	64	100	2	
$ZZ,X	74	116	2	
$HHLL,X	9E	158	3	
TAX	AA	170	1	N,Z
TAY	A8	168	1	N,Z
+ TRB $HHLL	1C	28	3	Z
$ZZ	14	20	2	
+ TSB $HHLL	0C	12	3	Z
$ZZ	04	4	2	
TSX	BA	186	1	N,Z
TXA	8A	138	1	N,Z
TXS	9A	154	1	---
TYA	98	152	1	N,Z

Tabelle A.5.2: **DEZ-/HEX-Zahlen und zugehörige 6502-Befehle**

DEZ-Zahl #	HEX-Zahl $	Befehl-Mnemonic	DEZ-Zahl #	HEX-Zahl $	Befehl-Mnemonic
0	00	BRK	32	20	JSR $HHLL
1	01	ORA ($ZZ,X)	33	21	AND ($ZZ,X)
2	02		34	22	
3	03		35	23	
4	04	+ TSB $ZZ	36	24	BIT $ZZ
5	05	ORA $ZZ	37	25	AND $ZZ
6	06	ASL $ZZ	38	26	ROL $ZZ
7	07		39	27	
8	08	PHP	40	28	PLP
9	09	ORA #$KK	41	29	AND #$KK
10	0A	ASL A	42	2A	ROL A
11	0B		43	2B	
12	0C	+ TSB $HHLL	44	2C	BIT $HHLL
13	0D	ORA $HHLL	45	2D	AND $HHLL
14	0E	ASL $HHLL	46	2E	ROL $HHLL
15	0F		47	2F	
16	10	BPL	48	30	BMI
17	11	ORA ($ZZ),Y	49	31	AND ($ZZ),Y
18	12	+ ORA ($ZZ)	50	32	+ AND ($ZZ)
19	13		51	33	
20	14	+ TRB $ZZ	52	34	+ BIT $ZZ,X
21	15	ORA $ZZ,X	53	35	AND $ZZ,X
22	16	ASL $ZZ,X	54	36	ROL $ZZ,X
23	17		55	37	
24	18	CLC	56	38	SEC
25	19	ORA $HHLL,Y	57	39	AND $HHLL,Y
26	1A	+ INC A	58	3A	+ DEC A
27	1B		59	3B	
28	1C	+ TRB $HHLL	60	3C	+ BIT $HHLL,X
29	1D	ORA $HHLL,X	61	3D	AND $HHLL,X
30	1E	ASL $HHLL,X	62	3E	ROL $HHLL,X
31	1F		63	3F	

DEZ-Zahl #	HEX-Zahl $	Befehl-Mnemonic	DEZ-Zahl #	HEX-Zahl $	Befehl-Mnemonic
64	40	RTI	96	60	RTS
65	41	EOR ($ZZ,X)	97	61	ADC ($ZZ,X)
66	42		98	62	
67	43		99	63	
68	44		100	64	+ STZ $ZZ
69	45	EOR $ZZ	101	65	ADC $ZZ
70	46	LSR $ZZ	102	66	ROR $ZZ
71	47		103	67	
72	48	PHA	104	68	PLA
73	49	EOR #$KK	105	69	ADC #$KK
74	4A	LSR A	106	6A	ROR A
75	4B		107	6B	
76	4C	JMP $HHLL	108	6C	JMP ($HHLL)
77	4D	EOR $HHLL	109	6D	ADC $HHLL
78	4E	LSR $HHLL	110	6E	ROR $HHLL
79	4F		111	6F	
80	50	BVC	112	70	BVS
81	51	EOR ($ZZ),Y	113	71	ADC ($ZZ),Y
82	52	+ EOR ($ZZ)	114	72	+ ADC ($ZZ)
83	53		115	73	
84	54		116	74	+ STZ $ZZ,X
85	55	EOR $ZZ,X	117	75	ADC $ZZ,X
86	56	LSR $ZZ,X	118	76	ROR $ZZ,X
87	57		119	77	
88	58	CLI	120	78	SEI
89	59	EOR $HHLL,Y	121	79	ADC $HHLL,Y
90	5A	+ PHY	122	7A	+ PLY
91	5B		123	7B	
92	5C		124	7C	+ JMP(HHLL,X)
93	5D	EOR $HHLL,X	125	7D	ADC $HHLL,X
94	5E	LSR $HHLL,X	126	7E	ROR $HHLL,X
95	5F		127	7F	

DEZ-Zahl #	HEX-Zahl $	Befehl-Mnemonic	DEZ-Zahl #	HEX-Zahl $	Befehl-Mnemonic
128	80	+ BRA	160	A0	LDY #$KK
129	81	STA ($ZZ,X)	161	A1	LDA ($ZZ,X)
130	82		162	A2	LDX #$KK
131	83		163	A3	
132	84	STY $ZZ	164	A4	LDY $ZZ
133	85	STA $ZZ	165	A5	LDA $ZZ
134	86	STX $ZZ	166	A6	LDX $ZZ
135	87		167	A7	
136	88	DEY	168	A8	TAY
137	89	BIT #$KK	169	A9	LDA #$KK
138	8A	TXA	170	AA	TAX
139	8B		171	AB	
140	8C	STY $HHLL	172	AC	LDY $HHLL
141	8D	STA $HHLL	173	AD	LDA $HHLL
142	8E	STX $HHLL	174	AE	LDX $HHLL
143	8F		175	AF	
144	90	BCC	176	B0	BCS
145	91	STA ($ZZ),Y	177	B1	LDA ($ZZ),Y
146	92	+ STA ($ZZ)	178	B2	+ LDA ($ZZ)
147	93		179	B3	
148	94	STY $ZZ,X	180	B4	LDY $ZZ,X
149	95	STA $ZZ,X	181	B5	LDA $ZZ,X
150	96	STX $ZZ,X	182	B6	LDX $ZZ,X
151	97		183	B7	
152	98	TYA	184	B8	CLV
153	99	STA $HHLL,Y	185	B9	LDA $HHLL,Y
154	9A	TXS	186	BA	TSX
155	9B		187	BB	
156	9C	+ STZ $HHLL	188	BC	LDY $HHLL,X
157	9D	STA $HHLL,X	189	BD	LDA $HHLL,X
158	9E	+ STZ $HHLL,X	190	BE	LDX $HHLL,Y
159	9F		191	BF	

DEZ-Zahl #	HEX-Zahl $	Befehl- Mnemonic	DEZ-Zahl #	HEX-Zahl $	Befehl- Mnemonic
192	C0	CPY #$KK	224	E0	CPX #$KK
193	C1	CMP ($ZZ,X)	225	E1	SBC ($ZZ,X)
194	C2		226	E2	
195	C3		227	E3	
196	C4	CPY $ZZ	228	E4	CPX $ZZ
197	C5	CMP $ZZ	229	E5	SBC $ZZ
198	C6	DEC $ZZ	230	E6	INC $ZZ
199	C7		231	E7	
200	C8	INY	232	E8	INX
201	C9	CMP #$KK	233	E9	SBC #$KK
202	CA	DEX	234	EA	NOP
203	CB		235	EB	
204	CC	CPY $HHLL	236	EC	CPX $HHLL
205	CD	CMP $HHLL	237	ED	SBC $HHLL
206	CE	DEC $HHLL	238	EE	INC $HHLL
207	CF		239	EF	
208	D0	BNE	240	F0	BEQ
209	D1	CMP ($ZZ),Y	241	F1	SBC ($ZZ),Y
210	D2	+ CMP ($ZZ)	242	F2	+ SBC ($ZZ)
211	D3		243	F3	
212	D4		244	F4	
213	D5	CMP $ZZ,X	245	F5	SBC $ZZ,X
214	D6	DEC $ZZ,X	246	F6	INC $ZZ,X
215	D7		247	F7	
216	D8	CLD	248	F8	SED
217	D9	CMP $HHLL,Y	249	F9	SBC $HHLL,Y
218	DA	+ PHX	250	FA	+ PLX
219	DB		251	FB	
220	DC		252	FC	
221	DD	CMP $HHLL,X	253	FD	SBC $HHLL,X
222	DE	DEC $HHLL,X	254	FE	INC $HHLL,X
223	DF		255	FF	

Hinweise zu den G65SC02-Zusatzbefehlen:

Der Befehl BRA ist vergleichbar mit den bekannten Branch-Befehlen BMI, BEQ etc. Der Unterschied hierbei ist nur, daß ohne jegliche Bedingung relativ gesprungen werden kann.
Mittels DEC läßt sich auch der Akku dekrementieren.
Mit PSX,PSY bzw. PLX,PLY können die Register X,Y ohne "Umweg" über den Akku direkt auf den Stack gelangen, bzw. von ihm zurückgeholt werden.
Der Befehl STZ speichert den Wert 0 in der adressierten Speicherzelle ab.
TSB führt die OR-Operation auf den Akku-Inhalt und die adressierte Speicherzelle M aus und legt das Ergebnis in M ab.
TRB führt die UND-Operation auf NOT A und die adressierte Speicherzelle M aus und legt das Ergebnis in M ab.
Es werden weiterhin zwei neue Adressierungsarten unterstützt:

- die Befehle ADC, AND, CMP, EOR, LDA, ORA, SBC, STA sind verknüpfbar mit der indirekten Zero-Page-Adressierung; symbolisch gekennzeichnet durch z.B. LDA ($ZZ). Es besteht hierbei volle Identität mit der indirekt indizierten Adressierung (Beispiel: LDA ($ZZ),Y) bei angenommenen Y-Wert 0.

- nur der Befehl JMP ($HHLL,X) weist die absolut indiziert indirekte Adressierung auf. Hierbei enthalten die Speicherzellen $HHLL+X,$HHLL+X+1 (L,H) die tatsächliche Sprungadresse.

Offsets von Basic-Befehlen und -Funktionen bzgl. Adresse $C09E

Zwischen den Adressen $C09E und $C19C sind in ASCII-Kodierung sämtliche VC-20-Befehle und -Funktionen hintereinander abgespeichert. Tabelle A.5.3 liefert eine Gegenüberstellung aller dieser Worte mit deren Offset bzgl. der Adresse $C09E.
Beispiel: das Wort LOAD weist gemäß Tabelle A.5.3 einen Offset von 4B auf, ist demnach ab der Adresse $C09E+4B=$C0E9 abgespeichert. Das Ende eines Wortes wird einfach daran erkannt, daß das Bit 7 des letzten Zeichens gesetzt ist. Im Beispiel "LOAD" stehen ab Adresse $C0E9 die HEX-Werte: 4C 4F 41 C4 (vgl. ASCII-Code-Tabellen im VC-20-Handbuch S. 145-147 oder Programmier-Handbuch S. 186). Der letzte Buchstabe "D" ist also nicht als 44 sondern mit gesetztem Bit 7 als C4 gespeichert.

END	00	ON	45	NEW	88	RND	C4
FOR	03	WAIT	47	TAB(	8B	LOG	C7
NEXT	06	LOAD	4B	TO	8F	EXP	CA
DATA	0A	SAVE	4F	FN	91	COS	CD
INPUT#	0E	VERIFY	53	SPC(	93	SIN	D0
INPUT	14	DEF	59	THEN	97	TAN	D3
DIM	19	POKE	5C	NOT	9B	ATN	D6
READ	1C	PRINT#	60	STEP	9E	PEEK	D9
LET	20	PRINT	66	AND	A7	LEN	DD
GOTO	23	CONT	6B	OR	AA	STR$	E0
RUN	27	LIST	6F	SGN	AF	VAL	E4
IF	2A	CLR	73	INT	B2	ASC	E7
RESTORE	2C	CMD	76	ABS	B5	CHR$	EA
GOSUB	33	SYS	79	USR	B8	LEFT$	EE
RETURN	38	OPEN	7C	FRE	BB	RIGHT$	F3
REM	3E	CLOSE	80	POS	BE	MID$	F9
STOP	41	GET	85	SQR	C1	GO	FD

Tabelle A.5.3

Betriebssystem-Fehlermeldungen

Die in Tabelle A.5.4 zusammengestellten Fehlermeldungen können in eigenen Programmen bei Bedarf mitbenutzt werden, wenn die entsprechende Fehlernummer im X-Register übergeben und zur dazugehörigen Betriebssystem-Routine $C43A = #50234 gesprungen wird.

Fehlernummer $	Fehlernummer #	Fehlermeldung
01	1	? TOO MANY FILES
02	2	? FILE OPEN
03	3	? FILE NOT OPEN
04	4	? FILE NOT FOUND
05	5	? DEVICE NOT PRESENT
06	6	? NOT INPUT FILE
07	7	? NOT OUTPUT FILE
08	8	? MISSING FILE NAME
09	9	? ILLEGAL DEVICE NUMBER
0A	10	? NEXT WITHOUT FOR
0B	11	? SYNTAX
0C	12	? RETURN WITHOUT GOSUB
0D	13	? OUT OF DATA
0E	14	? ILLEGAL QUANTITY
0F	15	? OVERFLOW
10	16	? OUT OF MEMORY
11	17	? UNDEF'D STATEMENT
12	18	? BAD SUBSCRIPT
13	19	? REDIM'D ARRAY
14	20	? DIVISION BY ZERO
15	21	? ILLEGAL DIRECT
16	22	? TYPE MISMATCH
17	23	? STRING TOO LONG
18	24	? FILE DATA
19	25	? FORMULA TOO COMPLEX
1A	26	? CAN'T CONTINUE
1B	27	? UNDEF'D FUNCTION
1C	28	? VERIFY
1D	29	? LOAD
1E	30	? BREAK

Tabelle A.5.4

LOAD/SAVE-Betriebssystem-Meldungen

Die in Tabelle A.5.5 aufgelisteten Meldungen können in eigenen Programmen benutzt werden:

Offset $	Offset #	Meldung
00	0	I/O ERROR #
0C	12	SEARCHING
17	23	FOR
1B	27	PRESS PLAY ON TAPE
2E	46	PRESS RECORD & PLAY ON TAPE
49	73	LOADING
51	81	SAVING
59	89	VERIFYING
63	99	FOUND
6A	106	OK

Tabelle A.5.5

Zwei Befehle sind hierzu notwendig:

```
im M-PGM:       LDY #$ ($-Offset)
                JSR $F1E2

im B-PGM:       POKE 782,(#-Offset):SYS 61922
```

Die hier angegebenen Anweisungen für Basic-Programme sind nur der Vollständigkeit halber angegeben, denn mittels PRINT-Anweisung ist die Ausgabe einer Meldung ja wesentlich einfacher.

Nr.	VIDEO-Speicher ab:	Farb-Speicher ab:	Inhalt: 648 = $0288	Inhalt: 36866 = $9002	Inhalt: 36869 = $9005
1	4096 = $1000	37888 = $9400	16	AND 127	AND 15 OR 192
2	4608 = $1200	38400 = $9600	18	AND 127 OR 128	AND 15 OR 192
3	5120 = $1400	37888 = $9400	20	AND 127	AND 15 OR 208
4	5632 = $1600	38400 = $9600	22	AND 127 OR 128	AND 15 OR 208
5	6144 = $1800	37888 = $9400	24	AND 127	AND 15 OR 224
6	6656 = $1A00	38400 = $9600	26	AND 127 OR 128	AND 15 OR 224
7	7168 = $1C00	37888 = $9400	28	AND 127	AND 15 OR 240
8	7680 = $1E00	38400 = $9600	30	AND 127 OR 128	AND 15 OR 240

Tabelle A.5.6

Die Tabelle A.5.6 gibt die Beziehung an zwischen den Video- und Farbspeichern sowie den Speicherzellen #648, #36866 und #36869 für 8 mögliche Fälle, die Sicht durch das "Fenster" in den VC-20 hinein sinnvoll zu variieren.

Die Benutzung dieser Tabelle wird anhand folgenden Beispiels einsichtig:
Nehmen wir an, Sie wollen den Bildschirmspeicher (Video-Speicher) ab Adresse #5632 = $1600 festlegen, dann geben Sie im Direkt- oder Programm-Modus ein (vgl. Nr. 4 in Tabelle A.5.6):

```
POKE 648,22:POKE 36866,PEEK(36866) AND 127 OR 128:
POKE 36869,PEEK(36869) AND 15 OR 208                (Return)
```

Der Farbspeicher wird hierbei (nicht von Ihnen beeinflußbar) ab Adresse #38400 = $9600 beginnen.

Die folgende Tabelle A.5.7 ist als Ergänzung zu A.5.6 anzusehen, um das Bit-Setzen bzw. -Löschen per AND- und OR-Operationen in obiger Zeile besser zu verstehen. Die Numerierung 1-8 ist sinngemäß mit Tabelle A.5.6 in Beziehung zu bringen.

Nr.	Zelle #36869 = $9005				Zelle #36866 = $9002
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 7
1	1	1	0	0	0
2	1	1	0	0	1
3	1	1	0	1	0
4	1	1	0	1	1
5	1	1	1	0	0
6	1	1	1	0	1
7	1	1	1	1	0
8	1	1	1	1	1

Tabelle A.5.7

Zeichengenerator-Speicher ab:		Zelle #36869 = $9005				Bits setzen mit:	
$	#	Bit 3	Bit 2	Bit 1	Bit 0	AND 240	OR
0000	0	1	0	0	0		8
0400	1024	1	0	0	1		9
0800	2048	1	0	1	0		10
0C00	3072	1	0	1	1		11
1000	4096	1	1	0	0		12
1400	5120	1	1	0	1		13
1800	6144	1	1	1	0		14
1C00	7168	1	1	1	1		15
8000	32768	0	0	0	0		0
8400	33792	0	0	0	1		1
8800	34816	0	0	1	0		2
8C00	35840	0	0	1	1		3
9000	36864	0	1	0	0		4
9400	37888	0	1	0	1		5
9800	38912	0	1	1	0		6
9C00	39936	0	1	1	1		7

Tabelle A.5.8

Die Tabelle A.5.8 zeigt den Zusammenhang zwischen dem Inhalt der ersten 4 Bits des VIC-Kontrollregisters CR5 (Adresse #36869) und der dadurch adressierten Anfangsadresse des Zeichenspeichers. Beispielsweise wird mittels:

```
POKE 36869,PEEK(36869) AND 240 OR 8        (Return)
```

der Zeichengenerator auf den Bereich $0000 bis $03FF für die normalen und $0400 bis $07FF für die invertierten (Cursor |) Zeichen eingestellt. Das VC-20 Betriebssystem "greift" sich also für das Zeichen "@" (s. Anhang H im VC-20-Handbuch) die ersten 8 Bytes, für "A" die zweiten 8 Bytes, etc., aus dem oben genannten Bereich heraus und stellt deren Inhalte in Form von Pixels auf dem Bildschirm dar. Natürlich werden Sie nach der Eingabe obigen Beispiels nicht besonders sinnvolle Zeichen zu sehen bekommen. Geben Sie dennoch die Buchstaben T und Y ein. Sie können somit ohne großen Aufwand die rasch sich verändernden Inhalte der Zellen $A0,A1,A3 (VC-20-Uhr) sowie $CD (Cursor-Blinkzähler) beobachten.

Eine weitere Anwendung der obigen Tabelle ist die Umschaltung von Großschrift/Graphik-Modus auf den Klein-/Großschrift-Modus bzw. umgekehrt.

	Einschalten des:
POKE 36869,PEEK(36869) AND 242 oder ?CHR$(14)	Klein-/Großschrift-Modus
POKE 36869,PEEK(36869) AND 240 oder ?CHR$(142)	Großschrift/Graphik-Modus

Inhalte der Zellen $C5=#197 und $CB=#203 bei gedrückter Taste

In so manchen Anwendungsfällen, z.B. in zahlreichen Spielen, ist es während des Programmablaufs nicht so sehr bedeutsam zu erkennen, welche Taste der Benutzer vor einiger Zeit eingegeben hat, sondern welche Taste just in dem Moment sich in gedrückter Stellung befindet, in dem diese Information auch vom Programm sofort ausgewertet wird. Außerdem soll hierbei auch nicht die Anzahl des Drückens dieser Taste, sondern vielmehr die gedrückte Stellung an sich von primärer Bedeutung sein.

Währenddessen die erstgenannte Möglichkeit durch den oft benutzten INPUT-Befehl realisiert wird (die gedrückten Tasten - auch die wiederholt gedrückten - werden im Tastaturpuffer zwischengespeichert), geschieht die Erkennung der zweiten Möglichkeit einfach durch Auslesen des Inhalts von Speicherzelle #197 oder #203 mittels PEEK(203).

Falls also z.B. an einer bestimmten Stelle eines Programms ein Sprung abhängig vom Drücken der F1-Taste gemacht werden soll, so ließe sich das einfach verwirklichen durch:

```
IF PEEK(203)=39 THEN ...
```

Nebenbei bemerkt, durch POKE 37154,0 wird die gesamte Tastatur gesperrt. In so manchen PGM'en kann das von Vorteil sein. POKE 37154,255 führt wieder den alten Zustand herbei.

#	Taste	#	Taste	#	Taste	#	Taste
0	1	16		32	SPACE	48	Q
1	3	17	A	33	Z	49	E
2	5	18	D	34	C	50	T
3	7	19	G	35	B	51	U
4	9	20	J	36	M	52	O
5	+	21	L	37	.	53	@
6	(Pfund)	22	;	38		54	(Pfeil oben)
7	DEL	23	CRSR RI	39	F1	55	F5
8	(Pfeil links)	24	STOP	40		56	2
9	W	25		41	S	57	4
10	R	26	X	42	F	58	6
11	Y	27	V	43	H	59	8
12	I	28	N	44	K	60	0
13	P	29	,	45	:	61	-
14	*	30	/	46	=	62	HOME
15	RETURN	31	CRSR DWN	47	F3	63	F7

64 keine Taste gedrückt

Tabelle A.5.9

Video-Interface-Chip (VIC 6561)

Register	Bits 7	6	5	4	3	2	1	0	Adressen $	#
CR0	I	SH = Horizontale Lage des Bildschirms							9000	36864
CR1	SV = Vertikale Lage des Bildschirms								9001	36865
CR2	V9	NL = Anzahl der Zeichen pro Zeile							9002	36866
CR3	R0	NR = Anzahl der Reihen						D	9003	36867
CR4	R8 - R1 = Zeilenzahl des Kathodenstrahls								9004	36868
CR5	/V13	V12	V11	V10	/C15	C12	C11	C10	9005	36869
CR6	LH = Horizontale Position des Lichtgriffels								9006	36870
CR7	LV = Vertikale Position des Lichtgriffels								9007	36871
CR8	PX = Horizontale Position des Paddle								9008	36872
CR9	PY = Vertikale Position des Paddle								9009	36873
CRA	SW	Tongenerator 1 (Tiefe Töne)							900A	36874
CRB	SW	Tongenerator 2 (Mittlere Töne)							900B	36875
CRC	SW	Tongenerator 3 (Hohe Töne)							900C	36876
CRD	SW	Tongenerator 4 (Geräusche)							900D	36877
CRE	AC = Hilfs-Farbe				Lautstärke				900E	36878
CRF	Hintergrundfarbe				B	Rahmenfarbe			900F	36879

Tabelle A.5.10

Erläuterungen zur Tabelle A.5.10

zu CR0: Der Ausgangswert von SH (Bits 0-6) ist #12. Die Verringerung (Vermehrung) um 1 entspricht einer Verschiebung des Bildschirms um 1/2 Zeichen = 4 Pixels nach links (rechts).
I (Bit 7) ist das sogenannte Interlace-Bit. Bei geeignetem Bildschirm und passendem Hardware-Zusatz soll es angeblich möglich sein, ein vom Sender oder vom Videorekorder herrührendes Bild in transparenter Weise mit Informationen vom VC-20 zu versehen.

zu CR1: Der Ausgangswert von SV ist #38. Die Verringerung (Vermehrung) um 1 entspricht einer Verschiebung des Bildschirms um 1/8 Zeichen = 2 Pixels nach oben (unten).

zu CR2: Der Ausganswert von NL (Bits 0-6) ist #22 (nämlich 22 Zeichen/Zeile).
V9 (Bit 7) ist Adreßbit 9 der Videospeicheranfangsadresse (weiteres siehe Tabelle A.5.7).

zu CR3: D (Bit 0) ist das Double-Bit. Wenn es auf 0 gesetzt ist, dann ist ein Zeichengröße als 8x8-Pixelmmatrix definiert. Ist es auf 1 gesetzt (dies geschieht z.B. mittels Eingabe von: POKE 36867,PEEK(36867) OR 1), dann wird jedes Zeichen als 8x16-Pixelmatrix dargestellt. Hierbei finden dann allerdings im gewohnten VC-20-Bildschirmrahmen nicht mehr 506, sondern nur noch die Hälfte = 253 Zeichen Platz. Zur besseren Verdeutlichung sei hier erwähnt, daß bei Ausgabe eines Zeichens auf dem Bildschirm zur dessen Darstellung 16 hintereinanderliegende Bytes (= 16 mal 8 Bits = 128 Pixels) im Zeichenspeicher, wo immer dieser auch liegen mag (siehe Tabelle A.5.8), "abgegriffen" werden. Im Falle von D=0 sind es folgerichtig nur 8 Bytes.
Der Ausgangswert von NR (Bits 1-6) ist binär 010111 = #23 (nämlich 23 Zeilen). Da jedoch diese Bitfolge ab Bit 1 beginnt, entspricht der Wert #23 bei Vernachlässigung der Bits 0 und 7 dem doppelten Wert, also #46.
R0 ist das niederwertigste Bit zu CR4.

zu CR4: Zusammen mit Bit 7 des CR3 als niederwertigstes Bit entspricht der dezimale Wert dieses Kontrollregisters der Nummer derjenigen Zeile, in der sich augenblicklich der Kathodenstrahl zur Darstellung der Pixels befindet.

zu CR5: Bits 0-3 sind die Adreßbits 10,11,12 und negiert-15 (der Schrägstrich in /C15 weist auf die negierte Bedeutung dieser Adreßleitung hin) der Zeichengeneratoranfangsadresse (siehe Tabelle A.5.8).
Bits 4-7 sind zusammen mit Bit 7 des CR2 die Adreßbits 9,10,11,12 und negiert-13 (der Schrägstrich in /V13 weist auf die negierte Bedeutung dieser Adreßleitung hin) der Videospeicheranfangsadresse (siehe Tabelle A.5.7).

zu CRA-CRD: SW (Bit 7) ist das Switch-Bit. Wenn es auf 1 gesetzt ist, werden die Töne gemäß der Inhalte der Bits 0-6 in der vorgegebenen Intensität (Bits 0-3 des CRE) an den Audio-Ausgang abgegeben.

zu CRE: Bits 0-3 beeinflussen die Lautstärke der auszugebenden Tone. Wert 0 heißt: keine Tonausgabe.
Bits 4-7 adressieren die Farben der Hilfs-Farbe gemäß Tabelle A.5.12. Die Hilfsfarbe wird dann benötigt, wenn ein Zeichen (bestehend aus einer 8x8-Pixelmatrix bei D=0 in CR3) nicht nur in einer einzigen Farbe gemäß Vorgabe der Bits 0-2 (die Farben 0-7 in Tabelle A.5.12) der zum Videospeicher zugehörigen Farbspeicherzellen, sondern in bis zu vier verschiedenen Farben dargestellt werden soll. Dieser sogenannte Mehrfarb-Modus wird durch Setzen des Bit 3 der jeweiligen Farbspeicherzelle eingeschaltet. Der Wert eines Bit-Paars (4 Bit-Paare gibt es in einer Pixelzeile, ein 8x8-Pixel-Zeichen besteht aus 8 Pixelzeilen) eines Zeichens adressiert die in diesen 2 Pixels gewünschte Farbe gemäß Tabelle A.5.11.

zu CRF: Bits 0-2 (Werte 0-7) gibt gemäß Tabelle A.5.12 die Farbe des Bildschirmrahmens an.
B (Bit 3) ist das Background-Bit, welches im Ausgangszustand gesetzt ist. Dies bedeutet das ein Zeichen mit der Vordergrundfarbe (Inhalt der Bits 0-2 der entsprechenden Farbspeicherzelle) auf einem Hintergrund der in Bits 4-7

angegebenen Farbe dargestellt wird. Ist B nicht gesetzt, so werden diese Farbverhältnisse einfach umgekehrt. Auf diese Weise kann man also erreichen, daß die Hintergrundfarben jedes einzelnen Zeichenplatzes verschieden sind. Übrigens tritt der in CRE besprochene Mehrfarb-Modus außerkraft, wenn B=0 ist.
Bits 4-7 adressieren die Hintergrundfarbe gemäß Tabelle A.5.12.

Bits	Farbe
00	Hintergrundfarbe (s. CRF)
01	Rahmenfarbe (s. CRF)
10	Vordergrundfarbe (Inhalt der Bits 0-2 des Farbspeichers)
11	Hilfs-Farbe

Tabelle A.5.11

Bits	$	Farbe
0000	0	Schwarz
0001	1	Weiß
0010	2	Rot
0011	3	Türkis
0100	4	Violett
0101	5	Grün
0110	6	Blau
0111	7	Gelb
1000	8	Orange
1001	9	Hellorange
1010	A	Rosa
1011	B	Helltürkis
1100	C	Hellviolett
1101	D	Hellgrün
1110	E	Hellblau
1111	F	Hellgelb

Tabelle A.5.12

Fachbücher von Markt & Technik

Software-Auswahl leicht gemacht 2. überarbeitete Auflage

1983, 423 Seiten, 2000 Programmbeschreibungen
Dieses Buch gibt Auskunft über Systemsoftware, branchenneutrale Anwendungssoftware, branchenorientierte Anwendungssoftware und technisch-wissenschaftliche Software in Form von Kurzbeschreibungen der einzelnen Softwarepakete. Aus allen Anwendungsbereichen für Personal Computer.

Best.-Nr. MT 340
DM 58,—*

Hardware-Auswahl leicht gemacht 2. überarbeitete Auflage

1982/83, 326 Seiten
Die wichtigsten Daten von über 200 Personal Computer-Systemen. Mit aktuellen Marktübersichten für Personal Computer sowie die wichtigsten Peripheriegeräte, mit einführenden Artikeln zu den verschiedenen Gerätetypen, Begriffserläuterungen, Auswahlkriterien (Checklisten), Trendberichten und Bezugsquellen. Eine Hilfestellung sowohl für den Computer-Einsteiger als auch für den »Profi«.

Best.-Nr. MT 350
DM 44,—*

K.-H. Heß

Basic-Programme für CBM/VC 20-Computer

1983, 150 Seiten
Die verschiedenen Aufgabenstellungen werden analysiert, allgemeingültige Lösungswege erarbeitet und in CBM-Basic konvertiert. Alle Programme sind ausführlich dokumentiert und anwendbar für die Serien CBM 2000, 3000, 4000 und 8000. Einige Programme laufen auch auf VC 20 und anderen basicprogrammierbaren Rechnern, wobei etwaige Programmanpassungen näher beschrieben sind.

Best.-Nr. MT 501
DM 32,—*

H.P. Blomeyer-Bartenstein

Personal Computer — das intelligente Werkzeug für jedermann

1983, 352 Seiten
Dieses Buch ist der Nachfolger des Standardwerks »Personal Computer — Kompaktrechner im Einsatz«. Es faßt den aktuellen Stand der Personal Computer-Technik zusammen.

Best.-Nr. MT 508
DM 53,—*

P. Rädsch

Programme und Tips für VC-20

1983, 152 Seiten
Anhand von nützlichen und unterhaltsamen Programmen können Sie mit diesem Buch die phantastischen und selten genutzten Möglichkeiten Ihres VC-20 nun voll ausnützen. Detaillierte Beispiele zeigen, wie Sie den Befehlswortschatz Ihres Home-Computers durch einfache Routinen verbessern können. Neben Spielprogrammen finden Sie u.a. auch Programme für Textverarbeitung, Rechnungsschreibung und Lagerverwaltung.

Best.-Nr. MT 513
DM 38,—*

P. Ewald

Software richtig eingekauft

1983, 144 Seiten
Informationen, Tips, Auswahlmethoden und Vorgehensweisen für alle, die sich Suche, Analyse, Leasing oder Kauf der richtigen Software erleichtern möchten

Best.-Nr. MT 505
DM 34,—*

*alle Preise inkl. MwSt. zuzügl. Versandkosten

Markt&Technik

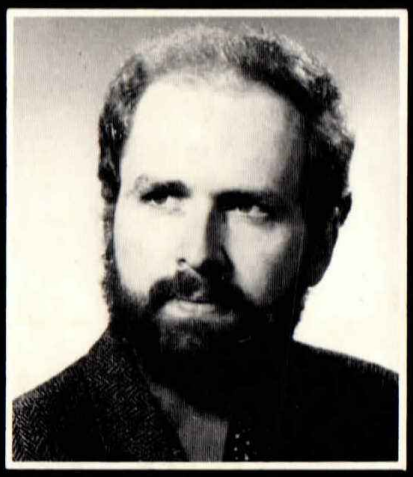
Hans-Pinsel-Straße 2 · 8013 Haar bei München · Telefon 089/4613-220

DAS VC-20 BUCH

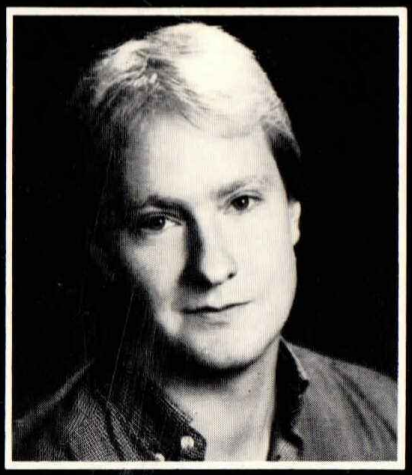
Dieses Buch soll dem VC-20-Benutzer in mehrerer Hinsicht nützlich sein:

- Es ist eine Sammlung von gut erklärten, mit nützlichen Kommentaren versehenen und zudem noch praktisch in vielerlei Weise einsetzbaren Programmen.
- Zum zweiten kann dieses Buch mit seinem umfassenden und sorgfältig zusammengestellten Anhang gut als Nachschlagewerk dienen.
- Eine weitere Richtung, die mit diesem Buch eingeschlagen wird, ist das noch bisher nur spärlich beleuchtete Thema »Maschinenprogramme speziell für den VC-20«. Dem Leser, auch wenn er sich nur auf diesem Gleis bewegen möchte, bieten sich hierfür zahlreiche auch auf andere Problemlösungen übertragbare, gut kommentierte Beispiele.

Dieses Buch zeigt dem Leser mannigfaltige Möglichkeiten dafür auf, daß sein VC-20 längst nicht nur als Spielcomputer, sondern auch für nützliche und kommerzielle Anwendungen im kleineren Rahmen einsetzbar ist. Beabsichtigt ist hiermit, nicht nur denjenigen Leserkreis anzusprechen, der bereits seit längerer Zeit mit dem VC-20 vertraut ist, sondern auch den, der erst frisch eingestiegen ist und nach kurzer Einarbeitungszeit sich mit vielen Fragen konfrontiert sieht. Mit gutem Recht kann behauptet werden, daß dieser in diesem Buch auf die meisten seiner Fragen eine Antwort zu finden vermag.



MICHAEL HEGENBARTH
geboren 1950, studierte Elektrotechnik an der TU Berlin. Nach mehreren Jahren Lehr- und Forschungstätigkeit als wissenschaftlicher Assistent im Fach »Theoretische Elektrotechnik«, widmet er sich seit 1980 den nationalen und internationalen Entwicklungen und Fragen der Teletex- und Bildschirmtext-Protokolle im FTZ der Deutschen Bundespost. Sein Hobby »Mikrocomputer« kommt ihm dabei sehr entgegen.



MICHAEL SCHÄFER
geboren 1956, kam zum gleichen Hobby über einen mehr praxisorientierten Weg. Er interessierte sich für die fernmeldetechnischen Anwendungen und wurde Fernmeldehandwerker bei der Deutschen Bundespost. Zur Zeit arbeitet er an der Entstörung und Wartung von Fernsprech-Nebenstellenanlagen. Sein Know-how in der Mikroprozessortechnik kommt der Weiterentwicklung dieser Geräte zugute.

Markt&Technik Verlagsgesellschaft mbH
ISBN 3-922120-50-4