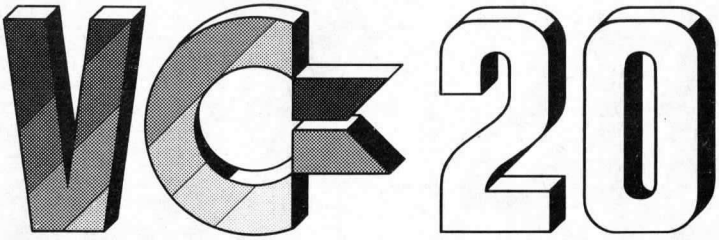
VC 20

VolksComputer

PROGRAMMIER HANDBUCH

commodore COMPUTER

VolksComputer



Programmier-Handbuch

commodore COMPUTER

Inhaltsverzeichnis Seite

Abbildungen

1.1 Einfuehrung

Dieses Handbuch geht davon aus, dass Sie als Anwender des VC20-Computers bereits ueber einige Grundkenntnisse der Programmiersprache BASIC verfuegen und nun die vielfaeltigen Programmierungsmoeglichkeiten des VC20 kennenlernen moechten.
Es handelt sich hierbei um ein vollstaendig in sich geschlossenes Programmierungshandbuch, das auch fuer den fortgeschrittenen Programmierer viele Anregungen und Hilfen bietet. Aber auch der Anfaenger kann unter Verwendung dieses Handbuches sehr bald Programme schreiben und auf dem VC20 ausfuehren.
VC20-BASIC ist eine leicht zu handhabende Programmiersprache, die die Welt der Farben und Toene in den Betrieb des Computers mit einbezieht.
Unter den vielen Eigenschaften des VC20 sind die normale und die hochaufloesende Farbgrafik sowie drei darstellbare Oktaven der Notenskala hervorzuheben. Natuerlich lassen sich auch beliebige Geraeusche zur Untermalung der im Handel erhaeltlichen oder selbstprogrammierten Video-Spiele erzeugen.
Sollten Sie in der Programmierung noch Anfaenger sein, so ist der VC20 der ideale Einstieg. Der einzige Weg, die Programmierung zu lernen, ist jedoch, einfach damit anzufangen. Unter Verwendung dieses Handbuches werden Sie sehr bald Ihre eigenen Programme schreiben koennen.
Zunaechst wollen wir jedoch einen Blick auf den Aufbau des VC20-Systems werfen.

1.2 Das VC20-System

Der Commodore-Video-Computer VC20 ist ein preiswerter Home-Computer, der in einem platzsparenden, leichten Gehaeuse untergebracht ist und an jedes Fernsehgeraet angeschlossen werden kann. Die vielfaeltigen Moeglichkeiten hinsichtlich der Farbsteuerung koennen natuerlich nur beim Anschluss an ein Farbfernsehgeraet oder einen Farbmonitor ausgeschoepft werden. Dabei wird ein Bildschirmformat von 22 Spalten zu 23 Zeilen erzeugt, d.h. es koennen bis zu 506 Zeichen auf dem Bildschirm abgebildet werden. Sowohl Farbe als auch Ton lassen sich beliebig steuern.
Das Herz Ihres VC20-Systems ist ein MCS6502-Mikroprozessor, der den Betrieb des Bildschirms, der Tastatur, der Kassettenstation sowie zusaetzlich anschliessbarer Peripheriegeraete vollstaendig steuert. Das Betriebssystem und der BASIC-Interpreter koennen nicht unbeabsichtigt zerstoert werden, da sie in ROM-Bausteinen (read only memory = nur lesbare Speicher) gespeichert sind.

1.2.1 Eigenschaften

Der VC20-Computer ist an jedes Farbfernsehgeraet oder an einen beliebigen Farbmonitor anschliessbar und hat folgende grundsaetzliche Eigenschaften:

* 5 kByte Anwenderspeicher (RAM), auf bis zu 32 kByte erweiterbar.
* Standard-CBM-BASIC.
* Farbe.

* Ton.
* Herausgefuehrte Peripherieanschluesse.
* Bildschirmanzeige 23 Zeilen/22 Spalten.
* Zeichensatz in hochaufgeloester Darstellung.
* Joystick, Ballschlaeger, Lichtstift.
* Stecksockel fuer Speichererweiterung und EPROM/ROM-Programme.

1.2.1.1 Farbe und Ton

Die Veraenderungen von Farbe und Ton werden beim VC20 durch das MSC6561-Video-Interface-Chip (VIC) gesteuert, das die Farbvideografik und die programmierbare Farbzeichengrafik mit hoher Bildschirmaufloesung erzeugt und es damit gestattet, System- und Video-Spiele aneinander anzupassen.

1.2.1.2 Spezielle Schnittstellen

Die RS-232-Schnittstelle (V24-Schnittstelle) des VC20 gestattet es unter anderem, diesen Computer ueber ein Datennetz mit anderen Computern zu koppeln und auf diese Weise Daten zwischen verschiedenen Systemen auszutauschen. Ausserdem lassen sich ueber spezielle Interface-Stecker beliebige, fuer den VC20 auf dem Markt erhaeltliche periphere Geraete (Drucker, Floppy-Disk) an die serielle Systemschnittstelle anschliessen. Das Systemblockschaltbild in Abb. 1.1 zeigt diese Moeglichkeiten in schematischer Form.

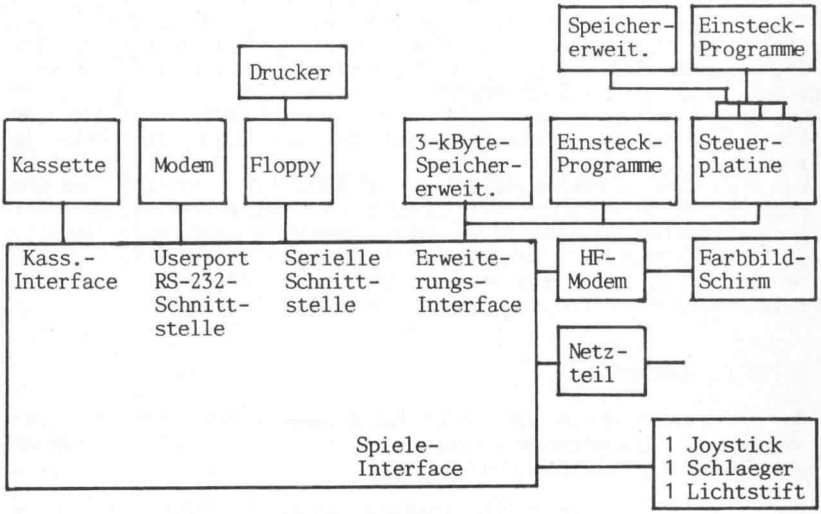


Abb. 1.1: Systemblockschaltbild des VC20

1.2.1.3 Speichererweiterung und Einsteckprogramme

Der Anwender- oder Arbeitsspeicher (RAM) des VC20 kann mit einfachen ROM- oder RAM-Einsteckeinheiten erweitert oder ergaenzt werden. Die RAM-Erweiterung kann in 3-, 8- oder 16-kByte-Schritten bis auf 32 kByte vorgenommen werden. Ausserdem sind eine ganze Reihe von Einsteckprogrammen zur Erweiterung der Programmiermoeglichkeiten erhaeltlich:

Supererweiterungskarte:

* 3 kByte RAM (macht den VC20 zum 8-kByte-Rechner).
* Hochaufloesende Grafik und PLOT-Befehle.
* Festbelegte Funktionstasten.
* Horizontales Bildschirmrollen.

Programmierhilfenkarte:

* TOOLKIT mit vielen Befehlen, die das Entwickeln komplizierterer BASIC-Programme erheblich erleich-tern.
* Maschinensprache-Monitor fuer den fortgeschrittenen Programmierer mit Systemkenntnissen.

2. Kommunikation mit dem VC20

2.1. Einfuehrung

Jede Kommunikation mit dem VC20 erfolgt ueber das Tastenfeld, das in Verbindung mit dem VC20-BASIC-Interpreter die notwendige Schnittstelle zwischen Ihnen als Anwender und dem Computer darstellt.
Der VC20 hat eine schreibmaschinenaehnliche Tastatur mit 62 Einzeltasten, die die Daten in ASCII-codierter Form dem Computer uebermittelt (ASCII = amerikanischer Standardcode fuer Informationsaustausch). Kapitel 4 dieses Handbuches enthaelt eine detaillierte Beschreibung der Funktionsweise der Tastatur. Abb. 2.1. zeigt das Tastenfeld des VC20.

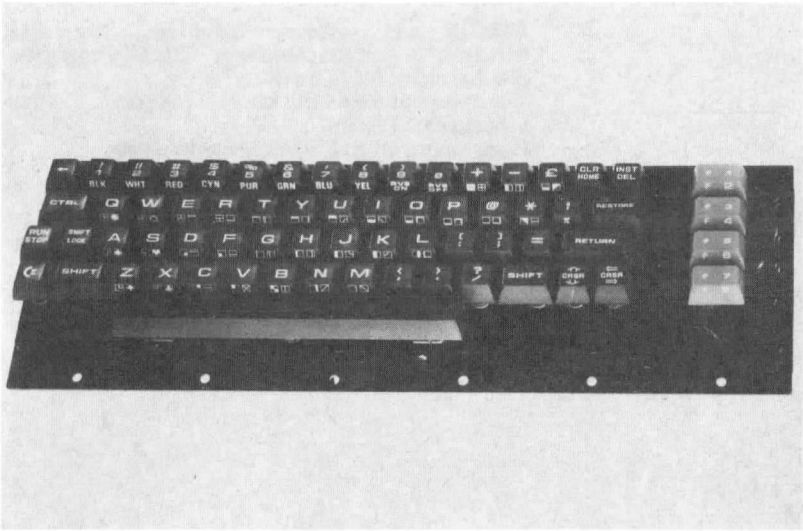


Abb. 2.1 Tastenfeld des VC20

2.2 Programmierte Befehle. Ein kurzer Ueberblick

Ehe wir mit dem Programmieren beginnen, wollen wir exakt beschreiben, was ein Programm ist, und was es tut.
Ein Programm besteht aus einer Reihe von Anweisungen, die der Computer ausfuehrt, um das vom Anwender gewuenschte Ergebnis zu liefern. Der Computer fuehrt diese programmierten Anweisungen Schritt fuer Schritt aus. Die Reihenfolge, in der die Anweisungen vom Computer ausgefuehrt werden, wird durch die Zeilennummern, die den Anweisungen vorangestellt werden, festgelegt. Die Zeilennummern bestimmen also zum einen die Reihenfolge, in der jeder Programmschritt abgearbeitet wird. Zum anderen bedeuten sie fuer den Computer, dass er die darauffolgende Information als Teil eines Programms im Speicher ablegt. Jede vom Anwender neu eingegebene Programmzeile wird in der richtigen numerischen Reihenfolge zwischen die schon abgespeicherten Programmzeilen eingefuegt. Wenn alle Programmzeilen abgespeichert sind, kann der Anwender dem Computer mitteilen, dass dieser das Programm ausfuehren soll. Wenn eine Anweisung dem Computer vom Anwender ohne eine vorangestellte Zeilennummer eingegeben wird, so wird diese Anweisung sofort ausgefuehrt und nicht als Programmzeile im Programmspeicher abgelegt.

2.3 Programmierung des VC20

Die folgenden Abschnitte (2.3 bis 2.4) wenden sich an den Anfaenger und an den VC20-Neuling. Wenn Sie genau den einzelnen Schritten folgen, werden Sie sehr schnell Erfolg haben. Wir wollen mit der Eingabe von Befehlen in den Computer beginnen.
Wie bereits erwaehnt, koennen dem Computer Befehle in zwei verschiedenen Modi eingegeben werden. Im Direkt-Modus und im Programm-Modus. Um Sie mit diesen beiden Modi vertraut zu machen, wollen wir ein wenig ueben.

2.3.1 Direkt-Modus

Im Direkt-Modus geben Sie den Befehl oder die Anweisung ueber die Tastatur ein und druecken zum Abschluss die RETURN-Taste. Der eingegebene Befehl wird dann sofort ausgefuehrt. Geben Sie folgendes ein:

```
PRINT 10-6
```

Druecken Sie die RETURN-Taste. Auf dem Bildschirm erscheint:

```
 4
```

Der Befehl (PRINT), den Sie eingegeben haben, wurde nach dem Druecken der RETURN-Taste sofort ausgefuehrt. Der BASIC-Interpreter uebersetzt den Befehl, der Computer berechnet den arithmetischen Ausdruck und das Ergebnis wird sofort angezeigt, in diesem Fall eine 4.
Nun geben Sie bitte folgendes ein:

```
PRINT 1/2,6*10
```

Druecken Sie die RETURN-Taste. Auf dem Bildschirm erscheint:

```
.5          60
```

Das /-Symbol bedeutet Division, das *-Symbol Multiplikation.
Sie haben jetzt ganz einfache Beispiele fuer Subtraktion, Division und Multiplikation unter Verwendung des PRINT-Befehls kennengelernt. Das Komma im zweiten Beispiel dient dazu, die Ergebnisse der beiden einzelnen arithmetischen Operationen (1/2 und 6*10) durch 10 Leerstellen voneinander getrennt anzuzeigen.
Der Direkt-Modus wird auch als Rechner-Modus bezeichnet, weil Sie in diesem Modus den VC20 wie einen Taschenrechner bedienen koennen. Betrachten Sie jetzt die folgenden Darstellungen:

```
?4.5+6.42              Addition
 10.92

READY.
?500-410               Subtraktion
 90

READY.
?3.4147*2              Multiplikation
 6.8294

READY.
?100/3                 Division
 33.3333333

READY.
?6/2*4-1               Gemischte Rechnung
 11
```

Im Direkt-Modus werden die Ergebnisse direkt in der naechsten Bildschirmzeile angezeigt. Um arithmetische Operationen im Direkt-Modus auszufuehren, muessen Sie die Zeile mit einem ? oder mit PRINT beginnen. Beide bedeuten "Drucke". Als naechstes geben Sie den arithmetischen Ausdruck, den Sie berechnet haben wollen, ein und druecken zum Abschluss die RETURN-Taste. Sie koennen aber auch im Direkt-Modus das Ergebnis einer Berechnung, die der Computer durch einen vorausgegangenen Befehl fuer Sie durchgefuehrt hat, anzeigen:

```
A=567.89*3

READY.
?A
 1703.67
```

Bei diesem Beispiel wurden dem VC20 im Direkt-Modus zwei verschiedene Anweisungen eingegeben. Wenn Sie die erste Anweisung, A=567.89*3, eingeben, wird das Ergebnis nicht angezeigt, da die Anweisung nicht durch ? oder PRINT eingeleitet wurde. Der VC20-BASIC-Interpreter hat jedoch die Berechnung ausgefuehrt und das Ergebnis einer Variablen mit Namen A zugewiesen. Durch die zweite Anweisung, ?A, wird der Inhalt der Variablen A, naemlich das Ergebnis der vorausgegangenen Berechnung, auf dem Bildschirm angezeigt.
Mit dieser Technik koennen Sie den Inhalt jeder Variablen, den man

als Wert der Variablen bezeichnet, anzeigen. Dabei muss die jeweilige Variable ihren Wert nicht durch die unmittelbar vorausgegangene Anweisung zugewiesen bekommen haben, wie dies in dem eben beschriebenen Beispiel der Fall war.

2.3.2 Programm-Modus

Im Programm-Modus beginnt jede Anweisung mit einer Zeilennummer. Die einzelnen Anweisungen bilden zusammen mit ihren Zeilennummern das Programm, das vom VC20 sequentiell, Zeile fuer Zeile, abgearbeitet wird.
Geben Sie die folgenden beiden Zeilen ein:

```
10 PRINT 7+8
20 PRINT 5-3
```

Denken Sie daran: Jede Zeile muss durch Druecken der RETURN-Taste abgeschlossen werden. Um das Programm zu starten, geben Sie jetzt

```
RUN
```

ein und druecken die RETURN-Taste. Das Programm beginnt mit der Ausfuehrung der Anweisung in der Zeile mit der niedrigsten Zeilennummer und beendet es mit der Zeile mit der hoechsten Zeilennummer. Auf dem Bildschirm wird jetzt also

```
15
2
```

angezeigt.
Bei der Eingabe der Programmzeilen ist es gleichgueltig, mit welcher Zeilennummer Sie beginnen. Der VC20-BASIC-Interpreter bringt sie auf jeden Fall in die richtige Reihenfolge.

2.3.3 Programmanzeige und Programmveraenderungen

Um das jeweils im Programm- oder Arbeitsspeicher befindliche Programm auf dem Bildschirm anzuzeigen, brauchen Sie nur

```
LIST
```

einzugeben und die RETURN-Taste zu druecken. Geben Sie also LIST ein und druecken Sie die RETURN-Taste. Auf dem Bildschirm erscheint:

```
LIST

10 PRINT 7+8
20 PRINT 5-3
READY.
```

Der Cursor (Blinker) verschwindet, waehrend das Programm gelistet wird und erscheint nach der READY.-Meldung wieder.
Um eine Programmzeile zu loeschen, geben Sie nur deren Zeilennummer ein und druecken die RETURN-Taste.

Also:

```
10<RETURN>
LIST
```

VC20 antwortet:

```
20 PRINT 5-3
READY.
```

Zeile 10 ist damit im Programm geloescht. Um sie wieder einzufuegen, gibt man wieder 10 und die gewuenschte Anweisung ein, abgeschlossen mit der RETURN-Taste.
Ein einfacherer Weg, Zeile 10 durch eine andere Anweisung zu ersetzen, ist, einfach die neue Zeile 10 einzugeben und mit RETURN abzuschliessen. Geben Sie also folgendes ein:

```
10 PRINT 8-7
LIST
```

VC20 antwortet:

```
10 PRINT 8-7
20 PRINT 5-3
READY.
```

Programmzeilennummern sollten nicht unmittelbar aufeinander folgen (1,2,3,4...). Es kann naemlich spaeter einmal notwendig werden, zwischen bereits existierende Programmzeilen neue Programmzeilen einzufuegen. Deshalb ist ein Zeilennummernabstand von 10 empfehlenswert.
Um ein ganzes Programm im Programmspeicher zu loeschen, geben Sie

```
NEW
```

ein und druecken die RETURN-Taste. Ehe Sie mit der Eingabe eines neuen Programms beginnen, sollten Sie mit NEW den Programmspeicher loeschen.

2.3.4 Anhalten des Programmablaufes

Sie moechten ggfs. Ihr Programm waehrend der Ausfuehrung anhalten. Dies wird Programmunterbrechung genannt und durch Druecken der Taste RUN/STOP waehrend des Programmablaufes ausgeloest. In einem solchen Fall gibt VC20 eine entsprechende Meldung zusammen mit der Nummer der Zeile aus, in der das Programm unterbrochen wurde. Also zum Beispiel:

```
BREAK IN 40
READY.
```

Da Ihr VC20 nur englisch "spricht", sind alle Meldungen, Anweisungen und Befehle in englischer Sprache abgefasst. BREAK heisst Unterbrechung.
Bei einer Unterbrechung kehrt VC20 in den Direkt-Modus zurueck, in dem Sie z. B. Direktanweisungen wie LIST, LOAD oder SAVE ausfuehren lassen koennen.

Um ein unterbrochenes Programm fortzusetzen, geben Sie

```
CONT
```

ein und druecken die RETURN-Taste. Das Programm wird dann exakt an der Stelle fortgesetzt, an der es unterbrochen wurde.

Jetzt haben Sie die ersten Grundkenntnisse der Programmierung Ihres VC20 erworben und wir koennen uns mit den vom System verwendeten Datenformaten beschaeftigen.

2.4 Zahlen und Daten-Formate

Die Zahlen, die der VC20 darstellen kann, unterliegen in ihrem Format gewissen Beschraenkungen. VC20 speichert alle Zahlen intern mit einer Genauigkeit von mehr als 9 Stellen. Auf dem Bildschirm werden jedoch immer nur hoechstens 9 Stellen angezeigt. Ausserdem koennen Zahlen auch mit einem Exponenten zur Basis 10 (wissenschaftliche Darstellung) dargestellt werden. Zu jedem Zeitpunkt, wenn Sie mit Ihrem VC20 arbeiten, werden Zahlen benoetigt. VC20 kann zwei verschiedene Zahlenarten speichern:

Gleitkommazahlen (Real-Zahlen)
Ganze Zahlen (Integer-Zahlen)

2.4.1 Gleitkommazahlen

VC20 stellt die Zahlen intern standardmaessig im Gleitkommaformat dar. Jede arithmetische Operation wird grundsaetzlich mit Gleitkommazahlen ausgefuehrt. Eine Gleitkommazahl kann eine ganze Zahl, eine gebrochene Zahl mit einem voranstehenden Dezimalpunkt (im Englischen wird anstelle des Dezimalkommas der Dezimalpunkt verwendet) oder eine Kombination aus beidem sein. Die Zahl kann ein positives (+) oder negatives (-) Vorzeichen haben. Ein positives Vorzeichen wird nicht mit ausgedruckt.
Betrachten wir einige Beispiele fuer Gleitkommazahlen:

Ganze Zahlen, die Integer-Zahlen entsprechen:

```
5
-15
6500
161
0
```

Gebrochene Zahlen mit einem Dezimalpunkt:

```
0.5
0.0165432
-0.00009
1.6
24.0085
-65.6
3.1416
```

Denken Sie daran, anstelle des Dezimalkommas den Dezimalpunkt zu verwenden, sonst erhalten Sie eine Fehlermeldung statt des gewuenschten Ergebnisses.

2.4.2 Rundung

VC20 bearbeitet alle Zahlen mit einer absoluten Genauigkeit von mindestens 8 Stellen, in einigen Faellen auch mit 9 Stellen, abhaengig von der Zahl. Alle zusaetzlichen signifikanten Stellen werden von VC20 aufgerundet, wenn die naechste Stelle 5 oder groesser ist und abgerundet, wenn sie 4 oder kleiner ist. Durch die Rundung kann es bei gebrochenen Zahlen zu Ungenauigkeiten in der 9. Stelle kommen:

```
?.555555556          Hier wird noch bei
 .55555555           der 6 ab-

?.555555557          und erst bei der 7
 .55555556           aufgerundet.

?.111111115          Hier wird noch bei
 .11111111           der 5 ab-

?.111111116          und erst bei der 6
 .11111112           aufgerundet.
```

Diese kleinen Ungenauigkeiten resultieren aus der Art und Weise, in der Computer Gleitkommazahlen intern abspeichern.

2.4.3 Wissenschaftliche Zahlendarstellung

Gleitkommazahlen koennen auch in der sogenannten wissenschaftlichen Zahlendarstellung ausgegeben werden. Wenn 10- und mehrstellige Zahlen eingegeben werden, werden diese vom VC20 automatisch in die wissenschaftliche Darstellung umgewandelt, sobald sie wieder ausgegeben werden sollen, um auf diese Weise auch grosse und sehr grosse Zahlen mit einer beschraenkten Stellenzahl relativ genau wiederzugeben. Zum Beispiel:

```
?1111111114
 1.11111111E+09

READY.
?1111111115
 1.11111112E+09
```

Eine Zahl in wissenschaftlicher Darstellung hat die allgemeine Form:

ZahlE±ee

Dabei bedeuten:

Zahl	eine ganze oder eine gebrochene Zahl, wie in Abschnitt 2.4.1 beschrieben.
E	der Buchstabe E fuer Exponent.
±	das Vorzeichen zum Exponenten.

ee — ein bis zu zweistelliger Exponent zur Basis 10. Dieser Exponent spezifiziert die Groesse der Gleitkommazahl, d.h. die Anzahl von Stellen, um die der Dezimalpunkt nach rechts (positiver Exponent) oder nach links (negativer Exponent) verschoben werden muss, um die tatsaechliche Dezimalpunktposition einzunehmen.

Hier einige Beispiele:

Wissenschaftliche Darstellung:	Standard-Darstellung:
2E1	20
10.5E+4	105000
66E+2	6600
66E-2	0.66
-66E-2	-0.66
1E-10	0.0000000001
94E20	9400000000000000000000

Wie vor allem die letzten beiden Beispiele zeigen, ist die wissenschaftliche Darstellung der geeignete Weg, sehr kleine oder sehr grosse Zahlen wiederzugeben. Das VC20-BASIC gibt alle Zahlen zwischen 0.01 und 999999999 in Standarddarstellung und alle Zahlen ausserhalb dieses Bereiches in wissenschaftlicher Darstellung wieder.
Versuchen Sie folgende Beispiele:

```
?.009
 9E-03

READY.
?.01
 .01

READY.
?99999998.9
 999999999

READY.
?999999999.6
 1E+09
```

Natuerlich kann Ihr VC20 nicht jede beliebige Zahlengroesse verarbeiten, selbst in wissenschaftlicher Darstellung nicht. Die Grenzen sind:

Groesste Gleitkommazahl: ±1.70141183E+38
Kleinste Gleitkommazahl: ±2.93873588E-39

Jede groessere Zahl fuehrt zu einem sogenannten Ueberlauf-Fehler (englisch: OVERFLOW ERROR).

Zum Beispiel:

```
?1.70141184E+38
?OVERFLOW ERROR
READY.
```

Jede kleinere Zahl wird auf Null abgerundet. Zum Beispiel:

```
?2.93873587E-39
 0

READY.
```

2.4.4 Ganze Zahlen

Eine ganze Zahl ist eine Zahl ohne Dezimalpunkt und ohne Dezimalstellen, die positiv (ohne Vorzeichen) oder negativ (-) sein kann. Ganze Zahlen duerfen beim VC20 nur im Bereich zwischen -32767 und 32767 liegen. Folgende Zahlen sind ganze Zahlen:

```
0
1
44
32699
-15
```

Jede ganze Zahl kann jedoch auch in Gleitkommadarstellung wiedergegeben werden, da die ganzen Zahlen eine Untermenge der Gleitkommazahlen sind. Wenn Sie mit ganzen Zahlen rechnen, so wandelt der VC20-BASIC-Interpreter diese Zahlen zunaechst in Gleitkommazahlen um, ehe die arithmetischen Operationen durchgefuehrt werden. Der wichtigste Unterschied zwischen Gleitkommazahlen und ganzen Zahlen besteht fuer den VC20-Anwender darin, dass ein Feld (s. Abschnitt 2.4.9) mit ganzen Zahlen erheblich weniger Speicherplatz benoetigt, naemlich 2 Byte fuer eine ganze Zahl gegenueber 5 Byte fuer eine Gleitkommazahl.

2.4.5. Zeichenketten (Strings)

Zeichenketten werden in der Computer-Fachsprache als Strings bezeichnet. Deshalb wird im folgenden in diesem Zusammenhang nur noch von Strings gesprochen.
Ein String besteht aus einem oder mehreren Zeichen, die von Anfuehrungszeichen (") am Anfang und Ende eingeschlossen sind.
Zum Beispiel:

```
"HALLO"
"VC20-RECHNER"
"12345"
"DER BETRAG IST DM 10.89"
```

Alle Datentasten (Buchstaben, Ziffern, Sonderzeichen, Grafiksymbole) mit Ausnahme der "-Taste sowie die drei Cursor-Steuertasten (CLR/HOME, Cursor aufwaerts/abwaerts Cursor links/rechts) und die RVS/OFF-Taste koennen zur Bildung von Strings benutzt werden.

Die einzigen Tasten, die nicht zur Bildung eines Strings verwendet werden koennen, sind die SHIFT-, die RUN/STOP-, die RETURN- und die INST/DEL-Tasten.
Alle Zeichen innerhalb eines Strings werden so abgebildet, wie sie im String stehen. Da jedoch den Cursor-Steuertasten sowie der RVS/OFF-Taste kein Standardzeichen entspricht, werden sie in einem String durch besondere Zeichen in inverser Darstellung (dunkles Zeichen auf hellem Grund) gekennzeichnet.
Wenn Sie einen String ueber das Tastenfeld eingeben, so kann dieser eine Laenge haben, die sich aus 88 minus aller anderen fuer die Programmzeile erforderlichen Zeichen (Zeilennummer, Anweisungen) ergibt. Wenn Sie also im Direkt-Modus einen String anzeigen lassen wollen, so koennen Sie bis zu 85 Stringzeichen (88 minus ? minus zweimal ") eingeben.
Der VC20-BASIC-Interpreter kann jedoch im Speicher Strings bis zu einer Laenge von 255 Zeichen ablegen. Solche Strings koennen Sie durch Aneinanderfuegen von mehreren Teilstrings erzeugen (s. Abschn. 3.2.11).

2.4.6. Variablen

Im Abschnitt 2.3.1 wurde bereits das Konzept der Variablen angedeutet. Dieses Konzept soll nun im folgenden eingehend beschrieben werden.
Eine Variable ist eine Dateneinheit, deren Wert (Inhalt) veraendert werden kann. Der Wert wird durch die Zahl bestimmt, die der Variablen zugewiesen wurde. Wenn Sie im Direkt-Modus

```
PRINT 10,20,30
```

eingeben, so antwortet VC20 mit

```
10          20          30
```

Wann immer Sie die obige Anweisung eingeben, erhalten Sie das darunter dargestellte Ergebnis, da bei der PRINT-Anweisung konstante Daten verwandt wurden. Sie koennen aber auch im Direkt-Modus

```
A=10:B=20:C=30:PRINT A,B,C
```

eingeben und erhalten

```
10          20          30
```

Zwar ist dies dasselbe Ergebnis wie oben; es wurden jedoch die Variablen A, B und C anstelle der Konstanten 10, 20 und 30 verwendet. Indem Sie die Werte, die Sie A, B und C zugewiesen haben, veraendern, veraendern Sie auch das von VC20 ausgegebene Ergebnis. Geben Sie ein:

```
A=-4:B=45:C=4E2:PRINT A,B,C
```

und Sie erhalten:

```
-4          45          400
```

Sie werden noch sehen, dass in nahezu allen Computer-Programmen Variablen verwendet werden.

2.4.6.1 Variablennamen

Variablen werden durch Namen identifiziert. In dem im letzten Abschnitt beschriebenen Beispiel haben wir die Variablennamen A, B und C verwendet. Eine Variable wird grundsaetzlich durch ihren Namen und durch ihren Wert gekennzeichnet. Der Variablenname repraesentiert einen Speicherbereich, in dem der gegenwaertige Wert gespeichert wird. In dem folgenden Beispiel sei der gegenwaertige Wert von A 14, von B 15 und von C 0, also:

Name	Speicherplatzinhalt
A	14
B	15
C	0

Unter Verwendung der Direkt-Modus-Anweisung

```
A=-1
```

wird der Inhalt der Speicherzellen, die durch die Variable A repraesentiert werden, von 14 auf -1 veraendert. Ein Variablenname repraesentiert also eine Speicheradresse, bei der der gegenwaertige Wert der Variablen gespeichert wird. Dabei ist eines sehr wichtig:

> Variablennamen, die ja der Programmierer vergibt, sind willkuerlich; es gibt keinen innewohnenden Zusammenhang zu dem Wert, den diese Namen repraesentieren.

Ein Variablenname kann aus beliebig vielen Zeichen bestehen. Das erste Zeichen muss aber ein Buchstabe zwischen A und Z sein; das zweite Zeichen kann entweder ebenfalls ein Buchstabe zwischen A und Z, eine Ziffer zwischen 0 und 9 oder % oder $ sein. Als letztes Zeichen des Namens darf wieder entweder ein Buchstabe oder eine Ziffer oder aber % oder $ stehen.

2.4.6.2 Gleitkommavariablen

Gleitkommavariablen repraesentieren Gleitkommazahlen. Dieser Variablentyp wird von Ihnen in Ihren Programmen vermutlich am haeufigsten verwendet werden. Folgende Namen duerfen Sie z.B. Gleitkommavariablen geben:

```
A
BERTA
A1
AA
Z55
```

2.4.6.3 Ganzzahl-Variablen (Integer-Variablen)

Ganzzahlvariablen repraesentieren ganze Zahlen. Namen fuer Ganzzahlvariablen muessen am Ende das %-Zeichen enthalten, wie die folgenden Beispiele zeigen:

```
A%
BERTA%
A1%
INTEGER%
X44%
```

Beachten Sie, dass auch Gleitkommavariablen ganze Zahlen repraesentieren koennen. Sie sollten jedoch gerade bei Feldern, die in Abschnitt 2.4.9 naeher beschrieben werden, wenn irgend moeglich Ganzzahlvariablen verwenden, die je Feldelement nur 2 Byte gegenueber 5 Byte bei Gleitkommavariablen benoetigen.

2.4.6.4 String-Variablen

Eine Stringvariable repraesentiert eine Kette beliebiger Zeichen (z.B. Text) und muss am Ende des Variablennamens ein $-Zeichen enthalten, wie die folgenden Beispiele zeigen:

```
A$
BERTA$
A1$
TEXT$
ZX$
X44$
```

2.4.6.5 Lange Variablennamen

Zur leichteren Lesbarkeit eines Programmes duerfen Sie auch, wie bereits erwaehnt, Variablennamen vergeben, die laenger als 2 alphanumerische Zeichen sind. Sie muessen jedoch dabei beachten, dass

1. nur die beiden ersten Zeichen signifikant sind, d.h. vom VC20-BASIC-Interpreter erkannt werden, und
2. bei String- oder Ganzzahl-Variablen das letzte Zeichen des Namens $ oder % sein muss.

Deshalb ist fuer den VC20-BASIC-Interpreter BANANE und BANDAGE derselbe Name, da nur BA als Name erkannt wird.
In einem Inventurprogramm ist natuerlich der Name TEILNR fuer eine Teilenummer aussagekraeftiger als der Name TN.
Das BASIC des VC20 erlaubt Variablennamen von bis zu 255 Zeichen Laenge. Die folgenden Beispiele sind Namen mit mehr als der Minimalzahl von Zeichen:

```
MITGLIEDER
T1234567
TEXTZEILE$
ABCDEF%
```

Zusammenfassend muessen Sie also bei der Vergabe laengerer Variablennamen folgendes beachten:

1. Nur die ersten beiden Zeichen sowie ggfs. das Identifikationszeichen (% oder $) sind signifikant.

2. Es gibt im VC20-BASIC "reservierte Woerter" (s. Abschn. 2.4.7). Es handelt sich dabei um Woerter, die fuer den VC20BASIC-Interpreter eine spezielle Bedeutung haben. Keines dieser reservierten Woerter darf an beliebiger Stelle eines von Ihnen vergebenen Namens vorkommen.

3. Die zusaetzlichen Zeichen in den Namen benoetigen zusaetzlichen Speicher, der vielleicht fuer Programmverlaengerungen dringender gebraucht wird.

2.4.7 Reservierte Woerter

Der VC20-BASIC-Interpreter erkennt einige Woerter im Programm als Aufrufe fuer spezielle Operationen. Die Namen, die fuer solche Aufrufe verwendet werden, werden als reservierte Woerter bezeichnet. Diese Woerter duerfen deshalb nicht als Variablennamen vergeben werden; auch nicht als Teil an irgend einer Stelle des Namens. Tabelle 2.1 enthaelt alle reservierten Woerter, von denen vor allem die 2-stelligen zu beachten sind, da diese am ehesten als Variablennamen verwendet werden koennten.

Tabelle 2.1: Reservierte Woerter

ABS	GET	NOT	SIN
AND	GET#	ON	SPC
ASC	GO	OPEN	SQR
ATN	GOSUB	OR	ST
CHR$	GOTO	PEEK	STEP
CLOSE	IF	POKE	STOP
CLR	INPUT	POS	STR$
CMD	INPUT#	PRINT	SYS
CONT	INT	PRINT#	TAB
COS	LEFT$	READ	TAN
DATA	LEN	REM	THEN
DEF	LET	RESTORE	TI
DIM	LIST	RETURN	TI$
END	LOAD	RIGHT$	TO
EXP	LOG	RND	USR
FN	MID$	RUN	VAL
FOR	NEW	SAVE	VERIFY
FRE	NEXT	SGN	WAIT

2.4.8 Mathematische Operatoren

Ein Operator ist ein spezielles Zeichen, das dem VC20-BASIC-Interpreter anzeigt, welche Operation mit den jeweils dabei angegebenen Variablen oder konstanten Daten (Termen) auszufuehren ist. Einer oder mehrere Operatoren in Verbindung mit einem oder mehreren Termen bilden einen Ausdruck.
Im VC20-BASIC gibt es:

* Arithmetische Operatoren
* Vergleichsoperatoren
* Boolesche Operatoren

2.4.8.1 Arithmetische Operatoren

Ein arithmetischer Operator definiert eine mit den benachbarten Termen auszufuehrende arithmetische Operation. Solche Operatoren werden grundsaetzlich mit Gleitkommazahlen ausgefuehrt. Ganze Zahlen werden fuer diesen Zweck deshalb vorher in Gleitkommazahlen umgewandelt und nach der Operation wieder zurueck in Integer-Zahlen uebersetzt. Arithmetische Operationen und ihre Symbole sind:

Addition (+):

Das '+'-Zeichen besagt, dass der Term zur Linken zu dem Term zur Rechten des Symbols addiert werden soll. Bei numerischen Termen ist dies die direkte Addition. Z.B.:

```
2+2
A+B+C
X%+1
BR+10E-2
```

Das '+'-Zeichen kann auch bei Stringoperationen verwendet werden. Hier dient es jedoch zur Verkettung oder Aneinanderfuegung von Teilstrings. Der Unterschied zwischen der numerischen Addition und der Verkettung von Strings wird durch folgende Beispiele deutlich:

```
Numerische Addition:   3+4=7
String-Addition:       "BRIEF"+"BOGEN"="BRIEFBOGEN"
```

Durch die Stringverkettung lassen sich Strings mit einer Laenge bis zu 255 Zeichen erzeugen. Z.B.:

```
"VOR"+"WAERTS" ergibt "VORWAERTS"
"HAL"+"LO"     ergibt "HALLO"
A$+B$
"1"+CH$+E$
```

Subtraktion (-):

Das '-'-Zeichen besagt, dass der Term zur Rechten des Symbols vom Term zur Linken abgezogen werden soll. Z.B.:

```
4-1          ergibt 3
100-64       ergibt 36
A-B
55-142       ergibt -87
```

Das '-'-Zeichen kann auch als Vorzeichen (Negation) verwendet werden, also z.B.:

```
-5
-9E4
-B
4--2         dasselbe wie 4+2
```

Multiplikation (*):

Ein '*' besagt, dass der Term zur Rechten des Symbols mit dem Term zur Linken multipliziert werden soll. Z.B.:

```
100*2        ergibt 200
50*0         ergibt 0
A*X1
R%*14
```

Division (/):

Ein '/' besagt, dass der Term zur Linken des Symbols durch den Term zur Rechten dividiert werden soll. Z.B.:

```
10/2         ergibt 5
6400/4       ergibt 1600
A/B
4E2/XR
```

Potenzierung (↑):

Der Term zur Rechten (Exponent) des '↑'-Symbols gibt den Grad an, um den der Term zur Linken potenziert werden soll. Der Exponent kann jede Zahl, Variable oder jeder beliebige Ausdruck sein, solange das Ergebnis der Potenzierung innerhalb des erlaubten Zahlenbereiches des VC20 liegt. Z.B.:

```
2↑2          ergibt 4
12↑2         ergibt 144
A↑(B*2)
5↑(2↑A)
```

Wenn ein arithmetischer Ausdruck mehrere Operationen enthaelt, wie z.B.

```
A+C*10/2↑2,
```

so wird dieser Ausdruck vom VC20-BASIC-Interpreter in einer festgelegten Reihenfolge (Hierarchie) abgearbeitet. Zuerst wird potenziert (↑), dann werden ggfs. vorhandene negative Vorzeichen beruecksichtigt. Es folgen Multiplikation (*) sowie Division (/) und dann Addition (+) sowie Subtraktion (-). Operatoren derselben Hierarchie (*,/ und +,-) werden von links nach rechts abgearbeitet. Diese festgelegte Hierarchie kann durch die Verwendung von Klammern aufgehoben werden. Operationen innerhalb von Klammern werden grundsaetzlich zuerst und zwar in der oben angegebenen Hierarchie durchgefuehrt. Z.B.:

```
4+1*2              ergibt   6
(4+1)*2            ergibt  10
100*4/2-1          ergibt 199
100*(4/2-1)        ergibt 100
100*(4/(2-1))      ergibt 400
```

Wie das letzte Beispiel zeigt, koennen Klammern auch geschachtelt werden. In einem solchen Fall wird der Ausdruck in der innersten Klammer zuerst ermittelt, dann der in der naechstinneren usw.

2.4.8.3 Vergleichsoperatoren

Ein Vergleichsoperator spezifiziert eine "wahr"- oder "unwahr"-Bedingung zwischen ihm benachbarten Termen. Der durch den Operator festgelegte Vergleich wird ausgefuehrt und der bezogene Ausdruck wird dann entsprechend dem Vergleichsergebnis durch einen Wert von -1 bei "wahr" und von 0 bei "unwahr" ersetzt. Tabelle 2.2 auf der naechsten Seite enthaelt neben den bereits beschriebenen mathematischen Operatoren alle Vergleichsoperatoren sowie die im naechsten Abschnitt beschriebenen Booleschen Operatoren. Die Vergleichsoperatoren liegen in der Rechenhierarchie unter den arithmetischen Operatoren, d.h., Ausdruecke mit Vergleichsoperatoren werden nach den arithmetischen Ausdruecken ermittelt. Z.B.:

```
1=5-4      ist wahr (-1)
14>66      ist unwahr (0)
15>=15     ist wahr (-1)
A<>B
```

Vergleichsoperatoren koennen auch beim Vergleich von Strings verwendet werden. In diesem Fall haben die Buchstaben des Alphabetes die Ordnung A<B<C<D... Strings werden verglichen, indem ihre gespeicherten Zeichenwerte, also ihre ASCII-Codes (s. Anhang E) miteinander verglichen werden. Z.B.:

```
"A"<"B"    ist wahr (-1)
"X"="XX"   ist unwahr (0)
C$=A$+B$
```

Tabelle 2.2: Mathematische Operatoren

Prioritaet hoch	Operator	Bedeutung
9	()	Aufhebung der Hierarchie
8	↑	Potenzierung
7	-	Negation
6	*	Multiplikation
6	/	Division
5	+	Addition
5	-	Subtraktion
4	=	Gleich
4	<>	Ungleich
4	<	Kleiner als
4	>	Groesser als
4	<=,=<	Kleiner oder gleich
4	>=,=>	Groesser oder gleich
3	NOT	Logisches Komplement
2	AND	Logisches UND
1	OR	Logisches ODER

niedrig

Die Ziffern in der Prioritaet-Spalte dienen nur zu Kennzeichnung der hierarchischen Ordnung. Sie haben sonst keine Bedeutung.

2.4.8.4 Boolesche Operatoren (logische Ausdruecke)

Die Booleschen Operatoren NOT, AND und OR spezifizieren eine logische Verknuepfung zwischen den beiden Termen rechts und links vom Operator. Im Fall von NOT wird nur der rechts stehende Term bearbeitet. In der Rechenhierarchie liegen die Booleschen Operatoren an niedrigster Stelle. Beispiele:

```
IF A=100 AND B=100 GOTO 10
```

Nur wenn sowohl A als auch B gleich 100 sind, verzweigt das Programm nach Zeile 10.

```
IF X<Y AND B>=44 THEN F=0
```

Der Variablen F wird nur dann der Wert 0 zugewiesen, wenn sowohl X kleiner als Y als auch B groesser oder gleich 44 ist.

```
IF A=100 OR B=100 GOTO 20
```

Das Programm verzweigt nach Zeile 20, wenn entweder A=100 ist oder B=100 ist.

```
IF X<Y OR B>=44 THEN F=0
```

Der Variablen F wird dann der Wert 0 zugewiesen, wenn entweder X kleiner als Y oder B groesser oder gleich 44 ist.

```
IF A=1 AND B=2 OR C=3 GOTO 30
```

Das Programm verzweigt nach Zeile 30 wenn entweder sowohl A=1 als auch B=2 ist, oder wenn C=3 ist.

Auch ein einzelner Term kann mit VC20-BASIC auf "wahr" oder "unwahr" geprueft werden, indem naemlich jeder von Null verschiedene Wert als wahr und ein Null-Wert als unwahr interpretiert wird. Z.B.:

`IF A THEN B=2` `IF A<>0 THEN B=2`	Beide Befehlszeilen sind gleichwertig.
`IF NOT B GOTO 100` `IF B=0 GOTO 100`	Auch hier sind beide Befehlszeilen gleichwertig, die zweite ist aber uebersichtlicher.

Die Booleschen Operationen lassen sich in einer Wahrheitstabelle zusammenfassen:

Tabelle 2.3: Boolesche Wahrheitstabelle

```
wahr   AND wahr   = wahr
unwahr AND wahr   = unwahr
wahr   AND unwahr = unwahr
unwahr AND unwahr = unwahr

wahr   OR  wahr   = wahr
unwahr OR  wahr   = wahr
wahr   OR  unwahr = wahr
unwahr OR  unwahr = unwahr

       NOT wahr   = unwahr
       NOT unwahr = wahr
```

2.4.8.5 Bit-orientierte Boolesche Operationen

Dieser Abschnitt ist fuer Anwender gedacht, die tiefer in die Computer-Mathematik eindringen wollen. Er beschaeftigt sich mit der Moeglichkeit, einzelne Bits in einem Byte zu setzen oder zu loeschen oder bestimmte Bitmuster in einem Byte zu pruefen. Wenn Sie das nicht interessiert, koennen Sie beruhigt bei Abschnitt 2.4.9 weiterlesen. Ihnen entgehen keine wichtigen Informationen.

Bei diesem Abschnitt werden Grundkenntnisse des Binaersystems sowie der hexadezimalen Schreibweise von Zahlen vorausgesetzt.
Bit-orientierte Boolesche Operationen werden normalerweise an vorzeichenlosen, positiven 16-Bit-Zahlen durchgefuehrt, also in einem Zahlenbereich zwischen $0000 und $FFFF (das $-Zeichen kennzeichnet die hexadezimale Schreibweise und hat nichts mit Strings zu tun) oder 0 und 65535 dezimal. Dem VC20-BASIC-Interpreter muessen die Zahlen, mit denen bit-orientierte Boolesche Operationen durchgefuehrt werden sollen, jedoch als Dezimalzahlen im Bereich zwischen -32767 und +32767 angegeben werden (s. a. Abschn. 3.2.10.4). Z.B:

Operation		Hexadezimales Aequivalent	
1 AND 1	ergibt 1	0001 AND 0001	ergibt 0001
1 AND -1	ergibt 1	0001 AND FFFF	ergibt 0001
15 OR 240	ergibt 255	000F OR 00F0	ergibt 00FF
NOT 0	ergibt -1	NOT 0000	ergibt FFFF
NOT 1	ergibt -2	NOT 0001	ergibt FFFE

Falls die zu bearbeitenden Terme noch nicht Integer-Zahlen sind, werden sie zunaechst in diese Form gebracht. Anschliessend wird jedes Bit des linken Terms entsprechend dem Booleschen Operator mit jedem Bit des rechten Terms logisch verknuepft. Das Ergebnis stellt wieder eine Integer-Zahl dar. Prinzipiell besteht kein Unterschied zwischen einer gemischten Booleschen Operation wie:

```
A=1 OR C<2
```

und einer einfachen Booleschen Operation wie:

```
A OR C
```

Der einzige praktische Unterschied ist der, dass im ersten Fall zunaechst die beiden Vergleichsoperationen ausgefuehrt werden, die 0 oder -1 liefern und dass dann die Boolesche Operation nur mit Werten von 0 und/oder -1 ausgefuehrt wird, waehrend im zweiten Fall A und C jede Integer-Zahl im Bereich zwischen -32767 und +32767 sein kann. Im Fall von:

```
IF A=B AND C<D GOTO 40
```

wuerden zunaechst die Vergleichsoperationen durchgefuehrt. Wenn die linke Operation "wahr" und die rechte "unwahr" ergibt, wird anschliessend die Boolesche Operation

```
IF -1 AND 0 GOTO 40
```

die das Ergebnis 0 liefert, also

```
IF 0 GOTO 40
```

Dies kann man, wie im letzten Abschnitt beschrieben, als

```
IF 0 <> 0 GOTO 40
```

schreiben, d.h. die Verzweigung wird nicht ausgefuehrt.

Eine an zwei Variablen durchgefuehrte Boolesche liefert eine Integer-Zahl:

```
IF A% AND B% GOTO 40
```

A% sei 255, B% sei 240. Dann ergibt sich aus obiger Boolescher Operation 240, d.h.

```
IF 240 GOTO 40
```

oder

```
IF 240<>0 GOTO 40
```

Die Verzweigung nach Programmzeile 40 wird also ausgefuehrt. Zum Abschluss noch zwei andere Beispiele:

```
A=A AND 10
A=A<10
```

Im ersten Fall wird der gegenwaertige Wert von A logisch mit 10 verknuepft und das Ergebnis wieder der Variablen A zugewiesen. A muss natuerlich eine Integer-Zahl im Bereich zwischen -32767 und +32767 sein. Im zweiten Beispiel wird die Vergleichsoperation A 10 ausgefuehrt, die 0 oder -1 ergibt, d.h. der Variablen A wird schliesslich entweder der Wert -1 oder 0 zugewiesen.

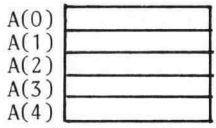
2.4.9 Felder

Ein Feld ist eine Folge von gleichartigen Variablen. Eine Zahlentabelle kann z.B. als Feld betrachtet werden. Die einzelnen Zahlen innerhalb dieser Tabelle werden dann zu Feldelementen. Wenn der gesamten Tabelle oder dem Feld ein Name zugeordnet wird, so kann das einzelne Feldelement durch seinen Platz in diesem Feld identifiziert werden.
Ein Feld kann eine Dimension, wie z.B. die erwaehnte Zahlentabelle haben, d.h., es handelt sich um eine Folge von Zahlen. Es kann aber auch mehrere Dimensionen haben. Zwei Dimensionen beschreiben eine Tabelle mit Zeilen und Spalten. Drei Dimensionen lassen sich als Wuerfel beschreiben oder als Stapel von zweidimensionalen Tabellen. Ab der 4. Dimension versagt unser Vorstellungsvermoegen. Mathematisch lassen sich solche Felder, die man auch Matrizen nennt, jedoch genauso einfach behandeln, wie zwei- oder dreidimensionale Felder.
Wir wollen nun die Felder genauer betrachten. Ein einzelnes Feldelement in einem eindimensionalen Feld hat im VC20-BASIC die allgemeine Form

```
Name(i)
```

wobei Name ein Variablenname getreu den in Abschnitt 2.4.6.1 beschriebenen Regeln ist. i gibt den Platz des Elementes in diesem Feld an.

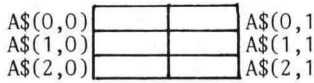
Ein eindimensionales Feld mit Namen A und 5 Elementen wird vom VC20-BASIC-Interpreter folgendermassen angelegt:



Die tatsaechliche Anzahl der Feldelemente ist gleich der Nummer des letzten Feldelementes plus 1, da der Interpreter immer mit Feldelement 0 beginnt.
Ein einzelnes Feldelement in einem zweidimensionalen Feld hat die allgemeine Form:

Name(i,j)

Name ist wieder ein Variablenname. i bezeichnet die Nummer der Zeile (Zeilenindex) und j die Nummer der Spalte (Spaltenindex). Ein zweidimensionales Feld mit Namen A$ mit 3 Zeilen und 2 Spalten wird vom VC20-BASIC-Interpreter folgendermassen angelegt:



Die Groesse eines solchen Feldes ergibt sich aus dem Produkt aus dem hoechsten Zeilenelement plus 1 und dem hoechsten Spaltenelement plus 1, in diesem Fall also 3*2=6 Elemente. Das VC20-BASIC erlaubt bis zu 255 Dimensionen. Das ist jedoch mehr ein theoretischer Wert. In den allermeisten Faellen werden Sie mit ein- oder zweidimensionalen Feldern auskommen.
Felder mit bis zu 11 Elementen (0 bis 10 bei einem eindimensionalen Feld) koennen ohne weitere Massnahmen wie normale Variablen verwendet werden. Erst wenn mehr als 11 Elemente benoetigt werden, muss das Feld mit Hilfe der DIM-Anweisung vorher dimensioniert werden. Diese Anweisung wird im Abschnitt 3.3.8 detailliert beschrieben. Eine Feldvariable, zu der ja immer ein in Klammern stehender Index gehoert, und eine gleichnamige einfache Variable werden vom VC20-BASIC-Interpreter als zwei verschiedene Variablen behandelt. Ist eine Variable einmal als Feld dimensioniert worden, kann sie nicht umdimensioniert werden.

2.5. BASIC-Funktionen

Ein wichtiger Bestandteil des VC20-BASIC sind die in BASIC integrierten Funktionen. Es sind dies:

* Arithmetische Funktionen
* String-Funktionen
* Format-Funktionen
* System-Funktionen

Jede dieser Funktionen fuehrt an einem spezifizierten Datenelement, das als Argument der Funktion bezeichnet wird, eine bestimmte Operation durch. Damit kann man z.B. die Quadratwurzel einer Zahl berechnen, man kann eine Gleitkommazahl in eine ganze Zahl umwandeln, man kann die Laenge eines Strings ermitteln, man kann Daten auf dem Bildschirm tabuliert ausgeben und vieles anderes mehr. Die BASIC-Funktionen erleichtern das Programmieren erheblich. Im folgenden geben wir einen Ueberblick ueber alle BASIC-Funktionen des VC20. Einzelheiten zu jeder Funktion entnehmen Sie bitte dem Kapitel 3. Das Argument einer Funktion kann aus einzelnen oder mehreren Konstanten, Variablen oder Ausdruecken bestehen. Die allgemeine Schreibweise fuer eine VC20-BASIC-Funktion ist:

Funktion(Argument1,Argument2,...)

Sind die einzelnen Argumente einer Funktion Ausdruecke, so werden diese vom BASIC-Interpreter zuerst ermittelt, so dass die Funktion selbst nur noch reine Zahlenwerte oder, im Falle von Stringfunktionen, Teilstrings verarbeitet. Wenn eine BASIC-Anweisung Funktionen enthaelt, so werden diese zuerst und dann erst der Rest der Anweisung ausgefuehrt.

Arithmetische Funktionen:

INT	Wandelt ein Gleitkommaargument in ein Integer-Argument.
SGN	Liefert das Vorzeichen des Argumentes. +1 fuer positives, -1 fuer negatives und 0 fuer Null-Argument.
ABS	Liefert den Absolutwert fuer ein Argument. Ein positives Argument wird nicht veraendert, ein negatives Argument wird in ein positives Argument umgewandelt.
SQR	Berechnet die Quadratwurzel des Argumentes.
EXP	Liefert die argument-fache Potenz der Zahl e, also e↑Argument.
LOG	Liefert den natuerlichen Logarithmus des Argumentes.
RND	Liefert eine Zufallszahl. Naeheres siehe Abschnitt 3.3.49.
SIN	Liefert den trigonometrischen SINUS des Argumentes, das im Bogenmass angegeben werden muss.
COS	Liefert den trigonometrischen COSINUS des Argumentes, das im Bogenmass angegeben werden muss.
TAN	Liefert den trigonometrischen TANGENS des Argumentes, das im Bogenmass angegeben werden muss.

ATN Liefert den trigonometrischen ARCUS-TANGENS des Argumentes, das im Bogenmass angegeben werden muss. Das Ergebnis ist ebenfalls ein Wert im Bogenmass.

Stringfunktionen:

STR$ Wandelt eine Zahl in eine entsprechende Ziffernzeichenkette.

VAL Wandelt eine Ziffernzeichenkette in eine entsprechende Zahl.

CHR$ Erzeugt aus einer Dezimalzahl das codeaequivalente ASCII-Zeichen.

ASC Erzeugt aus einem ASCII-Zeichen eine codeaequivalente Dezimalzahl.

LEN Liefert die Laenge des durch das Argument bezeichneten Strings.

LEFT$ Liefert den linken Teil des durch das Argument bezeichneten Strings. Das Argument enthaelt ausserdem die Anzahl der zu extrahierenden Zeichen.

RIGHT$ Liefert den rechten Teil des durch das Argument bezeichneten Strings. Das Argument enthaelt ausserdem die Anzahl der zu extrahierenden Zeichen.

MID$ Liefert einen Teilstring aus dem durch das Argument bezeichneten String. Das Argument enthaelt ausserdem den Anfang und die Anzahl der zu extrahierenden Zeichen.

Formatfunktionen:

SPC Erzeugt in Verbindung mit der PRINT-Anweisung soviele Leerstellen, wie das Argument angibt.

TAB Erzeugt in Verbindung mit der PRINT-Anweisung soviele Leerstellen, vom linken Zeilenanfang gerechnet, wie das Argument angibt.

POS Liefert in Verbindung mit einer PRINT-Anweisung die augenblickliche Cursor-Position in einer Bildschirmzeile. Das Argument ist ohne Bedeutung, muss aber angegeben werden (z.B. POS(0)).

Systemfunktionen:

PEEK Liefert den binaeren Inhalt einer Speicherzelle, deren Adresse durch das Argument bezeichnet wird.

TI$, TI — Liefern die durch eine Systemuhr erzeugt Zeit.

FRE — Liefert die Anzahl noch ungenutzter Bytes im Programmspeicher. Das Argument ist ohne Bedeutung, muss jedoch angegeben werden (z.B. FRE(0)).

ST — Liefert nach einer Ein/Ausgabeoperation das vom System gesetzte Statusbyte.

USR — Uebergibt die Programmsteuerung an ein Unterprogramm in Maschinensprache.

2.6 Farbregulierung beim VC20

Die Farbwiedergabe des Fernsehschirms oder Farbmonitors kann mit dem VC20 auf unterschiedliche Weise gesteuert werden. Entweder wird die Hintergrundfarbe des Bildschirms als eine einzelne Variable behandelt, in der dann die Farbinformationen sowohl fuer den Rahmen als auch fuer das Bildfenster zusammengefasst sind oder die Rahmen- und Bildfenster-Farbe werden getrennt voneinander geregelt. Die folgenden Abschnitte diskutieren diese Moeglichkeiten unter Verwendung der PRINT- und POKE-Anweisungen detailliert.

2.6.1 Hintergrundfarbe

Die Hintergrundfarbe des Bildschirms wird durch Veraendern des Inhaltes der Speicherzelle mit der Adresse $900F (dezimal 36879) gesteuert. In diese Zelle kann eine Zahl zwischen 1 und 255 unter Verwendung der POKE-Anweisung (s. Abschn. 3.3.23) gespeichert werden. Durch jede dieser Zahlen wird eine bestimmte Mischung aus Bildschirm und Rahmenfarbe festgelegt. Im folgenden Beispiel 1 wird die Hintergrundfarbe mit Zufallszahlen beliebig veraendert, waehrend die Farbe, in der die einzelnen Zeichen abgebildet werden, unveraendert bleibt. Einige Zahlen (die, bei denen das 4. Bit der Speicherzelle auf 1 gesetzt ist) liefern Zeichen in inverser Darstellung:

Beispiel 1:

```
10 R=INT(RND(1)*255)+1
20 C=36879
30 POKE C,R
40 FOR I=1 TO 200:NEXT:GOTO 10
```

Beispiel 2 zeigt, wie die einzelnen Bits der Zahl, die der Variablen R zugewiesen wird, gesteuert werden koennen, um entweder die Bildschirm- oder die Rahmenfarbe zu veraendern. In diesem Beispiel ist X eine Zahl zwischen 1 und 15, die die Bildschirmfarbe, und Y eine Zahl zwischen 0 und 7, die die Rahmenfarbe festlegt. Dabei ist zu beruecksichtigen, dass die hoeherwertigen 4 Bits der Speicherzelle mit der Adresse dezimal 36879 die Information fuer die Bildschirmfarbe und die niederwertigen 4 Bits die Information fuer die Rahmenfarbe bilden.

Beispiel 2:

```
10 C=36879
20 X=INT(RND(1)*15)
30 Y=INT(RND(1)*7)
40 R=X*16+Y+8
50 POKE C,R
60 FOR I=1 TO 200:NEXT:GOTO 20
```

Auch hier werden den Variablen X und Y Zufallszahlen zugewiesen. In Zeile 40 wird die Farbinformation aus Bildschirm- und Rahmenfarbe zusammengesetzt. Es sollte jeweils immer nur eine Farbinformation veraendert werden, waehrend die andere konstant bleibt. Die folgende Tabelle enthaelt die Farbsteuerzahlen fuer Bildschirm und Rahmen.

Tabelle 2.4: Farbsteuerzahlen

Bildschirm		Rahmen	
0	Schwarz	0	Schwarz
1	Weiss	1	Weiss
2	Rot	2	Rot
3	Blaugruen	3	Blaugruen
4	Purpur	4	Purpur
5	Gruen	5	Gruen
6	Blau	6	Blau
7	Gelb	7	Gelb
8	Orange		
9	Hellorange		
10	Rosa		
11	Hellblaugruen		
12	Hellpurpur		
13	Hellgruen		
14	Hellblau		
15	Hellgelb		

Beachten Sie bitte, dass die Bildschirm-Farbsteuerzahlen groesser als 8 hellere Farbtoene liefern, wodurch Zeichen in dunkleren Farbtoenen besser lesbar werden.

2.6.2 Zeichenfarben

Die Farbe, in der die Zeichen dargestellt werden koennen, laesst sich sowohl im Direkt- als auch im Programm-Modus leicht veraendern. Im Direkt-Modus brauchen Sie nur bei niedergehaltener CTRL-Taste eine der Tasten zwischen 1 und 8 in der obersten Reihe der Tastatur zu druecken. Der Cursor wird durch das so erzeugte Steuerzeichen auf die gewaehlte Farbe eingestellt, und zwar in derselben Reihenfolge, wie sie in der Tabelle 2.4 fuer die Rahmenfarben angegeben ist, jedoch jeweils um 1 erhoeht.
Im Programm-Modus werden die Zeichenfarben durch Drucken der Steuerzeichen mit Hilfe der PRINT-Anweisung veraendert. Alle Zeichenketten, denen ein solches Steuerzeichen vorangestellt wird, werden in der durch das Steuerzeichen festgelegten Farbe gedruckt. Im folgenden Beispiel 3 wird der Buchstabe A in zufaellig erzeugten Farben auf dem Bildschirm dargestellt.

Beispiel 3:

```
10 C$="<CTRL 1><CTRL 2><CTRL 3><CTRL 4>
      <CTRL 5><CTRL 6><CTRL 7><CTRL 8>"
20 C=INT(RND(1)*8)+1
30 PRINT MID$(C$,C,1)"A";
40 GOTO 20
```

Die Steuerzeichenkette in Zeile 10 wird beim Eingeben des Programms in Form von inversen Zeichen auf dem Bildschirm dargestellt. Durch Zeile 30 wird aus diesem String wahlfrei ein Steuerzeichen ausgewaehlt, das mit der PRINT-Anweisung in Zeile 20 die Farbe bestimmt, in der das A abgebildet wird.

2.6.2.1 Direktdarstellung von Zeichen auf dem Bildschirm

Eine andere Moeglichkeit, Zeichen auf dem Bildschirm darzustellen, besteht darin, Sie mit einem POKE-Befehl direkt in Zellen des Bildschirmspeichers (Videomatrix) abzulegen. Der Bildschirmspeicher beginnt bei der Adresse $1E00 (dez. 7680) und umfasst 506 Speicherzellen. Der Code, der zur Abbildung dieser Zeichen abgespeichert werden muss, kann aus dem CBM-Zeichencode hergeleitet werden, der dem ASCII-Code mit einer Ausnahme entspricht:
Dem ASCII-Code fuer kleine Buchstaben entsprechen beim CBM-Code die Grafik-Symbole. Um jetzt aus einem gegebenen CBM-Code, der ein 8-Bit-Code ist, den zugehoerigen Bildschirmcode zu erhalten, muss das Bit 6, also das zweithoechste Bit, dieses 8-Bit-Codes geloescht werden und durch Bit 7 ersetzt werden, falls dieses gesetzt ist.
Wenn also X% das Dezimalaequivalent des CBM-Codes ist, der in den Bildschirmcode umgewandelt werden soll, so kann das mit VC20-BASIC folgendermassen formuliert werden:

```
10 W%=X% AND 63
20 IF X% AND 128<>0 THEN W%=W% OR 64
30 X%=W%
```

In Zeile 10 wird das 6. Bit geloescht. In Zeile 20 wird Bit 6 gesetzt, falls Bit 7 gesetzt war. In Zeile 30 wird der Integervariablen X% der so erzeugte Bildschirmcode zugewiesen. Um diese Zeichen invers darzustellen, muss zu X% zum Schluss 128 addiert werden.
Der Bildschirmcode fuer den Buchstaben A ist 1. Durch die Anweisung

```
POKE 7680,1
```

wird also in der obersten linken Bildschirmecke, der sogenannten HOME-Position, ein A abgebildet, das jedoch unsichtbar bleibt, solange nicht der Cursor ueber dem A steht. Um das A sichtbar zu machen, muss in einer Referenz-Speicherzelle eine Farbinformation gespeichert werden. Jeder Bildschirmspeicherzelle ist naemlich eine Farbzelle zugeordnet, in der eine Farbinformation abgelegt werden kann. Die 506 Farbzellen beginnen bei der Adresse $9600 (dez. 38400). Wenn also ein rotes A in der HOME-Position des Bildschirms abgebildet werden soll, so muss der Befehl:

```
POKE 7680,1
```

gefolgt werden von

```
POKE 38400,2
```

Die Farbnummern sind hier dieselben wie fuer die Rahmenfarben (s. Tab. 2.4).

2.6.3 Beispielprogramme fuer die farbige Zeichendarstellung

Das Beispielprogramm zeigt, wie der ganze Bildschirm mit blauen Punkten beschrieben werden kann:

```
10 SL=7680:SC=38400
20 FOR I=0 TO 505
30 L=SL+I:C=SC+I
40 POKE L,81:POKE C,6
50 NEXT
```

Es ist zu beachten, dass im Verlauf der Abarbeitung der Schleife in den Programmzeilen 20 bis 50 bei jedem Schritt zur Anfangsadresse des Bildschirmspeichers und des Farbzellenspeichers derselbe Wert addiert wird, so dass die rechte untere Bildschirmecke die Adresse 7680+505 hat, der die Farbzelle mit der Adresse 38400+505 zugeordnet ist.

2.6.3.1 Auswahl von Farbkombinationen

Der VC20 erlaubt bis zu 255 Farbkombinationen fuer Zeichen-, Bildschirm- und Rahmenfarbe, die durch das folgende kleine Beispielprogramm erzeugt werden. Sie koennen den Lauf des Programms durch Druecken der STOP-Taste abbrechen. Die zu diesem Zeitpunkt dargestellte Farbkombination bleibt dann erhalten und in der linken oberen Bildschirmecke wird dann ihr Zahlenwert angegeben:

```
10 X=1
20 POKE 36879,X
30 PRINT "<CLR> POKE 36879,"X
40 X=X+1
50 FOR T=1 TO 1000:NEXT
60 GOTO 20
```

2.6.4 Adressaenderung bei Speichererweiterung

Eine wichtige Adressaenderung muessen Sie beachten, wenn Sie den Speicher Ihres VC20 ueber die Adresse $2000 (dez. 8192) hinaus erweitern. In diesem Fall wird naemlich die Adresse des Bildschirmspeichers automatisch auf $1000 (dez. 4096) und die des Farbzellenspeichers auf $9400 (dez. 37888) herabgesetzt.

2.7. Tonregulierung beim VC20

Sie koennen mit Ihrem VC20 annaehernd 3 Oktaven, Geraeusche und eine Vielfalt von Lautstaerken erzeugen. Die Tonfrequenz steigt mit der Groesse der Steuerzahl. Toene und Lautstaerke werden durch Veraendern der Inhalt von 5 Speicherzellen mit den Adressen $900A (dez. 36874), $900B (dez. 36875), $900C (dez. 36876), $900D (dez. 36877) und $900E (dez. 36878) geregelt. Die ersten drei Adressen dienen der Tonerzeugung, die vierte der Geraeuscherzeugung und die fuenfte der Lautstaerkenregelung, bei der nur die vier niederwertigen Bits ausgenutzt werden, so dass die Lautstaerke durch Zahlen zwischen 0 (leise) und 15 (laut) geregelt werden kann.

2.7.1 Erzeugung von Toenen

Bei VC20 werden Toene erzeugt, indem in die zugeordneten drei Speicherzellen mit der POKE-Anweisung Werte zwischen 128 und 255 abgelegt werden. Die Frequenz steigt mit der Zahl. Eine Ausnahme bildet 255. Dieser Wert ist einem niederfrequenten Ton zugeordnet. Jeder der drei Speicherzellen ist eine Stimmlage zugeordnet, die abgeschaltet wird, wenn die Speicherzelle eine Null enthaelt. Die in Tabelle 2.5 dargestellten Dezimalwerte erzeugen naeherungsweise 3 Oktaven der wohltemperierten Notenskala.

Tabelle 2.5: Zahlenaequivalente fuer Musiknoten

Note	POKE	Note	POKE
C	128	G	213
C#	134	G#	215
D	141	A	217
D#	147	A#	219
E	153	B	221
F	159	C	223
F#	164	C#	225
G	170	D	227
G#	174	D#	228
A	179	E	230
A#	183	F	231
B	187	F#	232
C	191	G	234
C#	195	G#	235
D	198	A	236
D#	201	A#	237
E	204	B	238
F	207	C	239
F#	210	C#	240

2.7.1.1 Beispielprogramm fuer die Tonerzeugung

Das folgende Beispielprogramm soll zeigen, welche Moeglichkeiten der VC20 bei der Tonerzeugung bietet:

```
10 A=36874
20 POKE A+4,15
30 FOR K=2 TO 10
40 FOR I=1 TO 10
50 POKE A,232+K*SIN(I)
60 FOR J=0 TO 100:NEXT J,I,K
70 POKE A+4,0
80 GOTO 10
```

2.8. Der Bildschirmeditor

Nachdem Sie bis jetzt einige Erfahrungen mit dem Abbilden und Veraendern von Zeichen auf dem Bildschirm sammeln konnten, wollen wir im folgenden die Eigenschaften des VC20-Bildschirmeditors im Detail beschreiben.
Kernstueck des Bildschirmeditors sind die Cursor-Steuertasten, mit denen der Cursor auf jede beliebige Position des Bildschirms gebracht werden kann, um Zeichen zu loeschen, einzufuegen oder zu ueberschreiben. Zeichen die bereits auf dem Bildschirm stehen, werden durch die Cursorbewegungen nicht veraendert. Die folgenden vier Funktionstasten dienen der Cursorsteuerung:

CLR/HOME
CRSR rauf/runter
CRSR links/rechts
INST/DEL

Jede dieser Tasten hat zwei Funktionen, abhaengig davon, ob die SHIFT-Taste mitgedrueckt wurde oder nicht. Die Funktion der letzten drei Tasten wird automatisch solange wiederholt, wie die jeweilige Taste niedergehalten wird. Anhand der im folgenden beschriebenen Schritte koennen Sie sich mit den Steuertasten und ihrer Funktion vertraut machen:

Schritt 1: Schalten Sie Ihren Computer an.

Schritt 2: Druecken Sie die CLR/HOME-Taste. Der Cursor wird in die linke obere Bildschirmecke gestellt.

Schritt 3: Druecken Sie bei niedergehaltener SHIFT-Taste die CLR/HOME-Taste. Der Bildschirm wird geloescht.

Schritt 4: Geben Sie jetzt das Alphabet ein. Beachten Sie, dass der Cursor, nachdem Sie das V eingegeben haben, auf den Anfang der naechsten Zeile wechselt, da jede Zeile nur 22 Zeichen aufnehmen kann. Wenn Sie beim Eingeben einen Fehler gemacht haben, so druecken Sie die INST/DEL-Taste, wodurch das falsche Zeichen geloescht wird, und geben dann weiter ein. Zum Schluss zeigen die ersten beiden Zeilen Ihres Bildschirms folgendes Bild:

```
ABCDEFGHIJKLMNOPQRSTUV
WXYZ
```

Schritt 5: Halten Sie die SHIFT-Taste nieder und druecken Sie die CRSR links/rechts-Taste (die Taste ganz rechts unten). Der Cursor wandert nach links und wechselt vom Anfang der zweiten auf das Ende der ersten Zeile. Am Ende der ersten Zeile lassen Sie die Taste los. Nun druecken Sie die INST/DEL-Taste solange, bis etwa die Haelfte der ersten Zeile geloescht ist und stellen dann den Cursor ueber den Buchstaben C:

```
ABCDEFGHIJKL
WXYZ
```

Schritt 6: Halten Sie die SHIFT-Taste nieder und druecken Sie die INST/DEL-Taste viermal. Es werden vier Leerstellen vor dem C eingefuegt und das C sowie der Text rechts vom Cursor werden um vier Stellen nach rechts verschoben:

```
AB    CDEFGHIJKL
WXYZ
```

Schritt 7: Schreiben Sie in diese Leerstellen vier Sterne:

```
AB****CDEFGHIJKL
WXYZ
```

Schritt 8: Druecken Sie die CRSR links/rechts-Taste solange, bis der Cursor auf den Anfang der zweiten Zeile wechselt und stellen Sie den Cursor rechts neben das Z:

```
AB****CDEFGHIJKL
WXYZ
```

Schritt 9: Loeschen Sie die vier Buchstaben in der zweiten Zeile durch viermaliges Druecken der INST/DEL-Taste:

```
AB****CDEFGHIJKL
```

Schritt 10: Loeschen Sie den Bildschirm mit der CLR/HOME-Taste bei gedrueckter SHIFT-Taste.

Schritt 11: Geben Sie ein:

```
DIES IST DIE ERSTE ZEILE
```

Dann druecken Sie bei niedergehaltener SHIFT-Taste die RETURN-Taste. Nun geben Sie ein:

```
DIES IST DIE ZWEITE ZEILE
```

und druecken die CLR/HOME-Taste.

Schritt 12: Druecken Sie die CRSR rauf/runter-Taste und zaehlen Sie jedes Druecken. Der Cursor wandert nach unten und erreicht nach dem 22. Druecken die unterste Bildschirmzeile.

Druecken Sie die CRSR rauf/runter-Taste ein weiteres Mal. Die oberste Textzeile verschwindet und die Zeile

DIES IST DIE ZWEITE ZEILE

steht nun in der ersten Bildschirmzeile. Dieses Einfuegen von Leerzeilen am unteren Bildschirmrand nennt man Aufrollen des Bildschirms. Der Bildschirm fasst also insgesamt 23 Zeilen zu je 22 Zeichen.

Schritt 13: Stellen Sie den Cursor auf die unterste Bildschirmzeile und geben Sie ein:

DIES IST DIE LETZTE ZEILE

Schritt 14: Druecken Sie jetzt die CRSR rauf/runter-Taste mehrmals. Die Textzeile wird bei jedem Druecken um eine Zeile nach oben verschoeben, der Bildschirm wird also aufgerollt. Druecken Sie nun die CLR/HOME-Taste und anschliessend bei niedergehaltener SHIFT-Taste die CRSR rauf/runter-Taste. Der Bildschirminhalt veraendert sich nicht. Es ist also nur ein Aufrollen, kein Abrollen moeglich.

2.8.1 Cursor-Steuermodi

Der Cursor kann sowohl direkt als auch programmiert gesteuert werden. Im Direkt-Modus wird der Cursor mit den im letzten Abschnitt beschriebenen Steuertasten ueber den Bildschirm gefuehrt, waehrend die Cursorbewegungen im Programm-Modus programmiert ausgefuehrt werden koennen. Die programmierte Cursor-Steuerung wird wiederum durch zwei Modi bewirkt, naemlich:

Anfuehrungsmodus
Einfuegungsmodus

Der Anfuehrungsmodus wird durch die Eingabe einer ungeraden Anzahl von Anfuehrungszeichen (") eingeschaltet. Danach wird der Cursor durch Druecken einer der Cursorsteuertasten nicht mehr bewegt, sondern der diese Funktion bewirkende Steuercode wird als ein invers dargestelltes Zeichen in die Zeile eingefuegt. Eine Ausnahme bilden hier die RETURN- und DEL-Taste. Der Anfuehrungsmodus wird durch ein weiteres Anfuehrungszeichen wieder ausgeschaltet.
Der Einfuegungsmodus wird durch Druecken der INST/DEL-Taste bei niedergehaltener SHIFT-Taste eingeschaltet und zwar fuer soviele Zeichen, so oft die INST/DEL-Taste gedrueckt wurde.
Diese beiden Modi koennen durch Druecken der RETURN- oder der RESTORE-Taste ausgeschaltet werden.
Mit den beschriebenen Steuertasten und Steuermodi koennen Sie so hilfreiche Editiermoeglichkeiten ausnutzen, wie:

* Einstellen eines Bildschirmfensters
* Loeschen ganzer Zeilen

* Erzeugen grafischer Darstellungen auf dem Bildschirm
* Modifizieren von Programmen durch Ueberschreiben aufgelisteter Programmzeilen.

2.8.2 Inverse Zeichendarstellung

Normalerweise wird auf dem Bildschirm ein helles Zeichen in der gewaehlten Farbe auf einem Hintergrund in einer anderen gewaehlten Farbe abgebildet. Diese Darstellung kann jedoch auch invertiert werden, d.h. die Zeichen erhalten die Hintergrund- und der Hintergrund erhaelt die Zeichenfarbe.
Versuchen Sie dazu folgendes Beispiel:

Schritt 1: Schalten Sie den Computer ein.

Schritt 2: Loeschen Sie den Bildschirm mit geshifteter CLR/HOME-Taste.

Schritt 3: Geben Sie AAA ein.

Schritt 4: Geben Sie bei niedergehaltener CTRL-Taste R ein.

Schritt 5: Geben Sie BBB ein. Die drei B's werden in der invertierten Darstellung wiedergegeben.

Schritt 6: Geben Sie bei niedergehaltener CTRL-Taste 0 ein. Der Bildschirm wird auf seine urspruenglich eingestellte Farbkombination zurueckgeschaltet.

Schritt 7: Pruefen Sie dies durch die Eingabe von CCC.

2.8.3 RUN/STOP-Taste

Bei niedergehaltener SHIFT-Taste bewirkt Druecken der RUN/STOP-Taste das Laden des ersten Programms von einer angeschlossenen Kassettenstation in den Programmspeicher des Rechners und den Start dieses Programms. Diese Funktion wird spaeter noch detailliert beschrieben.
Ungeshiftet hat diese Taste die Funktion einer Unterbrechungstaste. Wird sie waehrend der Ausfuehrung einer beliebigen Direktmodus-Anweisung oder eines Programms gedrueckt, so wird jede Aktivitaet des Computers unterbrochen und dieser wird in den Direkt-Modus zurueckgesetzt, indem die Meldung

READY.

auf dem Bildschirm erscheint und der blinkende Cursor auf neue Anweisungen wartet.

2.8.4 Zusammenstellung aller Steuertastenfunktionen

Im folgenden werden noch einmal alle Steuertasten fuer den Bildschirmeditor zusammen mit ihren Funktionen beschrieben:

Taste	ohne SHIFT	mit SHIFT
CLR/HOME	Setzt den Cursor in die linke obere Bildschirmecke.	Loescht den Bildschirm und setzt den Cursor in die linke obere Bildschirmecke
CRSR ob/un	Setzt den Cursor um eine Zeile nach unten. Wenn der Cursor vorher auf der untersten Bildschirmzeile stand, wird der Schirm um eine Zeile aufgerollt. Zeichen werden nicht beeinflusst. Hat Wiederholfunktion.	Setzt den Cursor um eine Zeile nach oben. Geht nicht ueber die obere Bildschirmbegrenzung hinaus. Zeichen werden nicht beeinflusst. Hat Wiederholfunktion.
CRSR li/re	Setzt den Cursor um eine Stelle nach rechts. Geht auf den Anfang der Folgezeile, wenn der Cursor vorher am Zeilenende stand. Zeichen werden nicht beeinflusst. Hat Wiederholfunktion.	Setzt den Cursor um eine Stelle nach links. Geht auf das Ende der vorhergehenden Zeile, wenn der Cursor vorher am Zeilenanfang stand. Zeichen werden nicht beeinflusst. Hat Wiederholfunktion.
INST/DEL	Loescht das Zeichen links neben dem Cursor. Alle Zeichen rechts vom geloeschten Zeichen werden um eine Stelle nach links versetzt. Hat Wiederholfunktion.	Fuegt an der Cursorposition eine Leerstelle ein. Alle Zeichen rechts davon werden um eine Stelle nach rechts versetzt. Hat Wiederholfunktion.
RETURN	Uebergibt die Information der Zeile links vom Cursor dem Interpreter und setzt den Cursor auf den Anfang der naechsten Zeile.	Setzt den Cursor auf den Anfang der naechsten Zeile, ohne die Information an den Interpreter zu uebergeben.
RUN/STOP	Unterbricht die augenblicklich vom Computer ausgefuehrte Anweisung und setzt den Computer in den Direkt-Modus.	Laedt und startet das erste Programm von Kassette.
SPACE	Druckt eine Leerstelle an der Cursorposition. Hat Wiederholfunktion.	Druckt eine geshiftete Leerstelle (nicht sichtbares grafisches Zeichen) an der Cursorposition.

2.8.5 Zusaetzliche Tastenfunktionen

Waehlen oder Aendern einer Farbe:

Bei niedergehaltener CTRL-Taste die Taste fuer die gewuenschte Farbe druecken.

Abbilden graphischer Symbole:

Bei niedergehaltener SHIFT-Taste die gewuenschte Zeichentaste druecken.

Wechsel des Zeichensatzes (Gross/Grafik oder Gross/Klein)

Bei niedergehaltener SHIFT-Taste die CBM-Taste (das ist die Taste ganz unten links aussen) druecken.

Erzeugen eines inversen Bildschirmhintergrundes:

Bei niedergehaltener CTRL-Taste die R-Taste druecken.

Erzeugen des normalen Bildschirmhintergrundes:

Bei niedergehaltener CTRL-Taste die 0-Taste druecken.

Abschalten verschiedener Modi (Einfuegungs-, Anfuehrungsmodus, hochaufloesende Grafik):

Bei niedergehaltener RUN/STOP-Taste die RESTORE-Taste druecken.

3. VC20-BASIC

3.1 Ueberblick

Im folgenden wird die VC20-Version des CBM-BASIC beschrieben. Sie lernen das Schreiben und die Syntax der BASIC-Anweisungen kennen. Jeder einzelne Befehl wird beschrieben und an kurzen Beispielen erlaeutert. Schliesslich wird die Datenein- und -ausgabe anhand von Beispielprogrammen detailliert beschrieben.

3.2 CBM-BASIC

3.2.1 Initialisierung des Rechners

Die Initialisierung des Rechners kann auf drei verschiedene Weisen erfolgen, die alle zur selben Bildschirmanzeige (s. Abb. 3.1) fuehren:

1. Einschalten des Rechners setzt das System in einen definierten Ausgangszustand und initialisiert die Variablen des BASIC-Interpreters und des Betriebssystems.

2. Das Ruecksetzsignal kann ohne Aus- und wieder Einschalten des Rechners durch einen Schalter am Speichererweiterungsanschluss generiert werden.

3. Durch einen Unterprogrammsprung (SYS) aus einem BASIC-Programm in die Initialisierungs-Routine des Betriebssystems.

Ein Warmstart, der das ggfs. im Speicher befindliche Programm nicht loescht, kann durch gleichzeitiges Druecken der STOP- und der RESTORE-Taste ausgeloest werden.

```
**** CBM BASIC V2 ****

3583 BYTES FREE

READY.
```

Abb. 3.1: Initialisierungsanzeige

3.2.2 Bedienungsmodi

Ihr VC20 kann von Ihnen in zwei verschiedenen Modi bedient werden. Im Direkt-Bedienungsmodus sind den eingegebenen BASIC-Anweisungen keine Zeilennummern vorangestellt, sondern sie werden so ausgefuehrt, wie sie eingegeben wurden. Die Ergebnisse arithmetischer und logischer Operationen koennen zwar sofort angezeigt und zur spaeteren Verwendung Variablen zugewiesen und damit gespeichert werden. Die Anweisungen selbst sind jedoch nach der Ausfuehrung verloren. Dieser Modus ist fuer die Fehlersuche unmittelbar nach einem Programmabbruch durch Fehlermeldung hilfreich. Auch einfache Probleme, fuer die kein Programm erforderlich ist, koennen in diesem Modus schnell geloest werden.
Im indirekten Bedienungsmodus werden Programme eingegeben. Den einzelnen Anweisungszeilen werden Nummern vorangestellt und die so zusammengesetzten Zeilen werden als Programmbestandteil nach aufsteigenden Zeilennummern geordnet im Speicher abgelegt. Ein so erfasstes Programm kann durch Eingabe der RUN-Anweisung gestartet werden.

3.2.3 Zeilenformat

BASIC-Programmzeilen haben das folgende allgemeine Format (die eckigen Klammern kennzeichnen wahlfreie Eintraege):

nnnnn BASIC-Anweisung[:BASIC-Anweisung..]<RETURN>

Nach Wahl des Programmierers koennen mehrere Anweisungen in eine Zeile geschrieben werden. Jede weitere Anweisung muss jedoch von der vorhergehenden durch einen Doppelpunkt getrennt werden. Eine BASIC-Programmzeile beginnt immer mit einer Zeilennummer (nnnnn), darf hoechstens 88 Zeichen (4 Bildschirmzeilen) enthalten und muss mit der RETURN-Taste abgeschlossen werden.

3.2.4 Zeilennummern

Jede BASIC-Programmzeile beginnt mit einer Zeilennummer. Die Zeilennummern legen die Reihenfolge fest, in der die Zeilen im Programmspeicher abgelegt werden und koennen ausserdem als Referenzen fuer Spruenge und beim Editieren dienen. Es sind nur ganzzahlige Zeilennummern zwischen 0 und 63999 erlaubt.

3.2.5 Zeichensatz des CBM-BASIC

Der CBM-BASIC-Zeichensatz besteht aus alphabetischen, numerischen und grafischen Zeichen sowie aus einigen Sonderzeichen. Die Tabelle auf der folgenden Seite gibt die vom VC20-BASIC akzeptierten Sonderzeichen und Cursor-Steuerzeichen wieder. Da letztere keine Zeichen im eigentlichen Sinne sind, sondern durch Tasten repraesentiert werden, werden sie generell in spitzen Klammern (<>) dargestellt.

Tabelle 3.1: Sonderzeichen und Cursor-Steuerzeichen

Zeichen	Name oder Funktion
	Leerstelle
;	Semikolon
=	Gleichheitszeichen oder Zuweisungssymbol
+	Plus-Zeichen oder Additionssymbol
-	Gedankenstrich oder Subtraktionssymbol
*	Stern oder Multiplikationssymbol
/	Schraegstrich oder Divisionssymbol
↑	Aufwaertspfeil oder Potenzierungssymbol
(	linke Klammer
)	rechte Klammer
%	Prozentzeichen
#	Nummernzeichen
$	Dollarzeichen
!	Ausrufungszeichen
[	linke eckige Klammer
]	rechte eckige Klammer
,	Komma
.	Punkt oder Dezimalpunkt
'	Apostroph
"	Anfuehrungszeichen
:	Doppelpunkt
&	kaufmaennisches Und
?	Fragezeichen oder PRINT-Symbol
<	Linke spitze Klammer oder "kleiner als"-Zeichen
>	Rechte spitze Klammer oder "groesser als"-Zeichen
\	Schraegstrich rueckwaerts
@	At-Zeichen
←	Pfeil links
£	Pfund-Zeichen
<DEL>	Loescht das zuletzt eingegebene Zeichen
<RETURN>	Beendet die Eingabe einer Zeile
<CRSR rechts>	Setzt den Cursor um eine Stelle nach rechts
<CRSR links>	Setzt den Cursor um eine Stelle nach links
<CRSR runter>	Setzt den Cursor um eine Zeile nach unten
<CRSR rauf>	Setzt den Cursor um eine Zeile nach oben
<CLR>	loescht den Bildschirm und setzt den Cursor in die HOME-Position
<HOME>	Setzt den Cursor in die HOME-Position
<INST>	Erlaubt das Einfuegen von Zeichen an der Cursorposition
<STOP>	Versetzt den Interpreter vom Programm- in den Direkt-Modus
<RUN>	Fuehrt die Befehlsfolge LOAD<RETURN> RUN<RETURN> aus
<CTRL>	Zusammen mit den Zifferntasten 1 bis 8 wird die gegenwaertige Bildschirmfarbe geaendert; zusammen mit R wird inverse Zeichendarstellung eingeschaltet
<RESTORE>	Zusammen mit <STOP> wird ein Warmstart des Systems ausgeloest
<CBM-Taste>	Wechselt den Zeichensatz (Gross/Klein oder Gross/Grafik)

3.2.6 Konstanten des CBM-BASIC

Konstanten sind aktuelle Werte, die der BASIC-Interpreter waehrend der Programmausfuehrung verwendet. Er unterscheidet dabei zwischen

String-Konstanten
numerische Konstanten

Eine Stringkonstante ist eine Folge von bis zu 255 alphanumerischen und/oder Sonderzeichen, eingeschlossen in Anfuehrungszeichen. Z.B.:

```
"HALLO"
"DM 22.50"
"HUND UND KATZE"
```

Numerische Konstanten sind positive oder negative Zahlen und duerfen keine Kommas sondern nur einen Punkt enthalten. BASIC unterscheidet zwei Typen von numerischen Konstanten:

Integer-Konstanten — Ganze Zahlen zwischen -32767 und +32767. Diese Zahlen duerfen keinen Dezimalpunkt enthalten.

Gleitkomma-Konstanten — Positive oder negative Zahlen, die intern in Exponentialdarstellung (wissenschaftliche Darstellung) gespeichert werden. Die Mantisse wird vom Buchstaben E und einem positiven oder negativen ganzzahligen Exponenten im Bereich zwischen -38 und +37 gefolgt. Es werden bis zu 9 signifikante Stellen ausgedruckt. Beispiele:

```
235.988E-4 = .0235988
2359E6     = 2359000000
```

3.2.7 Variablen

Variablen sind Namen, die Werte repraesentieren, welche in einem BASIC-Programm verwendet werden. Der Wert einer Variablen kann dieser explizit vom Programmierer oder, als Ergebnis von Berechnungen, vom Programm zugewiesen werden. Ehe einer Variablen ein Wert zugewiesen wird, wird ihr Wert vom Interpreter als Null im Fall von numerischen und als Leerstring (String der Laenge Null) im Fall von String-Variablen angenommen.

3.2.7.1 Variablennamen und -kennzeichnung

BASIC-Namen duerfen eine beliebige Laenge von bis zu 255 Zeichen haben. Fuer den Interpreter sind jedoch immer nur die ersten beiden und ggfs. das letzte Zeichen signifikant. Fuer Namen duerfen nur Buchstaben und Ziffern verwendet werden, wobei das erste Zeichen des Namens immer ein Buchstabe sein muss. Ausserdem werden die Variablentypen durch spezielle Zeichen gekennzeichnet (siehe nachfolgend).

Als Variablennamen duerfen keine reservierten Woerter (s. Abschn. 2.4.7) verwendet werden. Diese duerfen auch nicht eingebettet in Variablennamen vorkommen. Wenn ein Variablenname z. B. mit FN beginnt, betrachtet der BASIC-Interpreter dies als Aufruf einer vom Anwender definierten BASIC-Funktion. Variablen repraesentieren immer entweder einen numerischen Wert oder einen String (Zeichenkette).

3.2.7.2 Namen fuer Stringvariablen

Namen fuer Stringvariablen werden durch ein Dollar-Zeichen ($) auf der letzten Stelle gekennzeichnet. Z.B.:

```
A$="REGENSCHIRM"
```

Der Buchstabe A repraesentiert den String "REGENSCHIRM", das $-Zeichen bezeichnet den Variablentyp.

3.2.7.3 Namen fuer numerische Variablen

Numerische Variablen koennen durch ein Prozent-Zeichen (%) an der letzten Stelle des Namens als Integer-(Ganzzahl-)Variablen gekennzeichnet werden. Alle Namen ohne dieses Zeichen bezeichnen Gleitkommavariablen. Z.B.:

MI	Gleitkommavariable
GRENZWERT%	Integer-Variable

3.2.8 Feldvariablen

Ein Feld ist eine Gruppe oder Tabelle von Zahlen oder Strings, die durch einen Variablennamen repraesentiert wird. Jedes Feldelement wird mit einem Index zum Variablennamen beschrieben, der eine ganze Zahl oder ein Integer-Ausdruck sein kann. Ein Feldvariablenname hat genau soviele Indizes wie das Feld Dimensionen hat. Z.B.:

V(10)	Beschreibt einen Wert in einem eindimensionalen Feld.
T(1,4)	Beschreibt einen Wert in einem zweidimensionalen Feld.

Es sind theoretisch 255 Dimensionen moeglich.

3.2.9 Umwandlung von numerischen Variablen

Der VC20-BASIC-Interpreter erlaubt die Umwandlung numerischer Variabler von einem Typ in den andren. Dabei gelten folgende Regeln:

* Wenn eine numerische Konstante eines Typs einer numerischen Variablen eines anderen Typs zugewiesen wird, so wird die Konstante in dem Typ gespeichert, der dem der Variablen entspricht. Wird einer Stringvariablen

ein numerischer Wert zugewiesen oder umgekehrt, so meldet der Interpreter den Fehler

TYPE MISMATCH ERROR

Beispiel:

```
10 A%=23.42
20 PRINT A%
RUN
 23

READY.
```

* Alle arithmetischen und Vergleichsoperationen werden intern grundsaetzlich mit binaeren Gleitkommazahlen durchgefuehrt. Deshalb werden vor solchen Operationen mit Integerzahlen diese in das Gleitkommaformat und nach der Operation zurueck in das Integer-Format gewandelt.

* Logische Operatoren (s. Abschn. 3.2.10.4) konvertieren ihre Operanden in Integer-Zahlen und liefern Integer-Ergebnisse. Deshalb muessen die Operanden im Bereich zwischen -32767 und +32767 liegen. Andernfalls meldet der Interpreter den Fehler

  OVERFLOW ERROR

* Wenn ein Gleitkommawert in einen Integer-Wert umgewandelt wird, so wird ersterer grundsaetzlich auf die naechst kleinere Integer-Zahl abgerundet. Also z.B.:

```
10 A%=55.58:B%=-55.58
20 PRINT A%,B%
RUN
 55          -56

READY.
```

3.2.10 Ausdruecke und Operatoren

Ein Ausdruck kann einfach ein String oder eine numerische Konstante, Variable oder eine Kombination aus Konstanten und Variablen mit Operatoren zur Erzeugung eines einzelnen Wertes sein.
Operatoren fuehren mit Werten mathematische oder logische Operationen durch. Der VC20-BASIC-Interpreter kennt vier verschiedene Kategorien von Operatoren:

1. Arithmetische Operatoren
2. Vergleichsoperatoren
3. Logische Operatoren
4. Funktionsoperatoren

3.2.10.1 Arithmetische Operatoren

Der VC20-BASIC-Interpreter kennt folgende arithmetischen Operatoren in der Reihenfolge ihrer Beruecksichtigung:

Operator	Operation	Beispiel
↑	Potenzierung	X↑Y
-	Negation	-X
*,/	Multiplikation, Division	X*Y, X/Y
+,-	Addition, Subtraktion	X+Y, X-Y

Um diese Hierarchie aufzuheben koennen Klammern verwendet werden. Geklammerte Ausdruecke werden vom Interpreter grundsaetzlich zuerst ausgewertet. Werden mehrere Klammerebenen geschachtelt, so werden die Klammerausdruecke von innen nach aussen ausgewertet. Innerhalb eines Klammerpaares gilt die oben angegebene Hierarchie. Es duerfen nicht mehr als 10 Klammerebenen geschachtelt werden.
Die folgende Tabelle gibt die BASIC-Schreibweise von algebraischen Ausdruecken an:

Tabelle 3.2: BASIC-Schreibweise von algebraischen Ausdruecken

Algebraischer Ausdruck	BASIC-Schreibweise	Algebraischer Ausdruck	BASIC-Schreibweise
$X+2Y$	X+2*Y	$X-\frac{Y}{Z}$	X-Y/Z
$\frac{XY}{Z}$	X*Y/Z	$\frac{X+Y}{Z}$	(X+Y)/Z
$(X^2)^Y$	(X↑2)↑Y	X^{Y^Z}	X↑(Y↑Z)
X(-Y)	X*(-Y)	Zwei aufeinanderfolgende Operatoren muessen durch Klammerung getrennt werden.	

3.2.10.2 Ueberlauf und Division durch Null

Trifft der BASIC-Interpreter waehrend der Auswertung eines Ausdrucks auf eine Division durch Null, so wird die Auswertung mit der Fehlermeldung

DIVISION BY ZERO ERROR

abgebrochen.
Wird dagegen das Ergebnis einer Berechnung groesser als die maximal erlaubten Werte (±32767 bei Integer- und ±1.70141183E+38 bei Gleitkommazahlen), so reagiert der Interpreter mit folgenden Fehlermeldungen:

ILLEGAL QUANTITY ERROR	bei Integer-Zahlen
OVERFLOW ERROR	bei Gleitkomma-Zahlen

Wird das Ergebnis einer Berechnung kleiner als der minimal erlaubte Wert von ±2.93873588E-39, so wird auf Null abgerundet.

3.2.10.3 Vergleichsoperatoren

Vergleichsoperatoren dienen dem Vergleich von zwei Werten. Das Ergebnis ist entweder "wahr" mit dem Wert -1 oder "unwahr" mit dem Wert 0. Das Ergebnis kann dann in Verbindung mit der BASIC-Anweisung IF (s. Abschn. 3.3.14 zur Steuerung des Programmablaufes verwendet werden.

Operator	Vergleich auf	Beispiel
=	Gleichheit	X=Y
<>	Ungleichheit	X<>Y
<	Kleiner als	X<Y
>	Groesser als	X>Y
<=	Kleiner gleich als	X<=Y
>=	Groesser gleich als	X>=Y

Das Gleichheitszeichen wird ausserdem zur Zuweisung von Werten zu Variablen benutzt. Siehe dazu Abschnitt 3.3.17.
Wenn in einem Ausdruck sowohl arithmetische als auch Vergleichsoperatoren vorkommen, so werden zuerst die arithmetischen Operatoren abgearbeitet. Z.B.:

```
X+Y<(T-1)/Z
```

Dieser Ausdruck ist wahr, wenn der Wert von X+Y kleiner ist als der Wert von T-1 dividiert durch Z. Weitere Beispiele:

```
IF SIN(X)<0 GOTO 1000
IF I-INT(I/J)<>0 THEN K=K+1
```

3.2.10.4 Logische Operatoren

Logische Operatoren dienen zum Testen von Mehrfachvergleichen, zur Bit-Manipulation oder zum Durchfuehren Boolescher Operationen. Ein logischer Operator liefert ein bitweises Ergebnis, das entweder "wahr" (ein von Null verschiedener Wert) oder "unwahr" (Null) ist. In einem gemischten Ausdruck werden die logischen Operationen nach den arithmetischen und den Vergleichsoperationen durchgefuehrt. Im folgenden werden die Wahrheitswerte, die die drei logischen Operatoren NOT, AND und OR liefern, hierarchisch angegeben:

	Argument 1	Argument 2	Ergebnis
NOT	wahr	-	unwahr
	unwahr	-	wahr
AND	wahr	wahr	wahr
	wahr	unwahr	unwahr
	unwahr	wahr	unwahr
	unwahr	unwahr	unwahr

	Argument 1	Argument 2	Ergebnis
OR	wahr	wahr	wahr
	wahr	unwahr	wahr
	unwahr	wahr	wahr
	unwahr	unwahr	unwahr

Genau wie die Vergleichsoperatoren ueber ihr Ergebnis zur Steuerung des Programmablaufes beitragen koennen, kann dies auch durch die Verknuepfung von zwei oder mehreren Vergleichen durch logische Operatoren geschehen, die ja wiederum "wahr"- oder "unwahr"-Werte liefert (s. a. Abschn. 3.3.14). Beispiele:

```
IF D<200 AND F<4 THEN 80
IF I>10 OR K<0 THEN 50
IF NOT P THEN 100
```

Logische Operatoren arbeiten intern folgendermassen:
Zunaechst werden die beiden Operanden in ganze, vorzeichenbehaftete Zweierkomplement-16-Bit-Zahlen im Bereich zwischen -32767 und +32767 umgewandelt. Sind die Operanden groesser oder kleiner, so wird eine Fehlermeldung ausgegeben. Sind die beiden Operanden 0 und/oder -1, so liefert eine logische Operation ebenfalls 0 oder -1. Die logische Operation wird auf jeden Fall bitweise durchgefuehrt, d.h. jedes Ergebnisbit wird durch die entsprechenden Bits in den beiden Operanden bestimmt. Dadurch ist es moeglich, mit den logischen Operatoren das Bitmuster von Speicherzellen zu testen. Z. B. kann das Statusbyte an einer Ein/Ausgabe-Schnittstelle maskiert werden, um den Zustand eines bestimmten Bits zu testen (s. a. Abschn. 3.4.21). Auch kann mit dem OR-Operator in einem bestimmten Byte ein ganz bestimmtes Bitmuster erzeugt werden. Die folgenden Beispiele sollen die Arbeitsweise der logischen Operatoren erlaeutern:

```
63 AND 16 = 16          0000000000111111   63
                  AND   0000000000010000   16
                        0000000000010000 = 16

15 AND 14 = 14          0000000000001111   15
                  AND   0000000000001110   14
                        0000000000001110 = 14

-1 AND 8 = 8            1111111111111111   -1
                  AND   0000000000001000    8
                        0000000000001000 =  8

4 OR 2 = 6              0000000000000100    4
                  OR    0000000000000010    2
                        0000000000000110    6

10 OR 10 = 10           0000000000001010   10
                  OR    0000000000001010   10
                        0000000000001010 = 10

-1 OR -2 = -1           1111111111111111   -1
                  OR    1111111111111110   -2
                        1111111111111111 = -1
```

```
NOT 1 = -2                    0000000000000001    1
                        NOT   1111111111111110 = -2
```

Der Operator NOT bildet das Einerkomplement des Operanden

3.2.10.5 Funktionsoperatoren

Eine BASIC-Funktion in einem Ausdruck wird dazu verwandt, an einem Operanden eines festgelegte Operation auszufuehren. Solche im BASIC-Interpreter des VC20 integrierten Funktionen sind z. B. SQR (Quadratwurzel ziehen) oder SIN (trigonometrischer SINUS). Diese Funktionen werden in den Abschnitten 3.3.2 ff. detailliert beschrieben.

3.2.11 String-Operationen

Strings (Zeichenketten) koennen mit dem '+'-Zeichen miteinander verkettet werden, also z. B.:

```
10 A$="REGEN":B$="SCHIRM"
20PRINT A$+B$+"E"
RUN
REGENSCHIRME

READY.
```

Strings koennen ausserdem mit den bereits beschriebenen Vergleichsoperatoren miteinander verglichen werden. Dabei wird der Vergleich zeichenweise mit den ASCII-Codes der jeweiligen Zeichen durchgefuehrt. Sind alle ASCII-Codes der miteinander verglichenen Strings gleich, so sind auch die Strings gleich. Bei unterschiedlichen Codes steht der kleinere vor dem groesseren Code. Wenn waehrend des Vergleichs das Ende eines der beiden Strings erreicht wird, so gilt der kuerzere als der kleinere String. Vor- oder nachlaufende Leerstellen gehoeren mit zum String und sind daher signifikant. Z. B.:

```
"AA" < "AB"
"WOLKE" = "WOLKE"
"X$" > "X#"
"CL " > "CL"
"kg" < "KG"
"RAST" < "RASTE"
B$ < "9.12.78"  wobei B$ = "8.12.78"
```

3.2.12 Editieren von Programmzeilen

Wenn beim Editieren (Eingeben) von Programmzeilen ein unkorrektes Zeichen mit eingegeben wurde, so kann dieses einfach durch Druecken der DEL-Taste geloescht werden und die Zeile kann weiter eingegeben werden. Um ein bereits im Speicher befindliches Programm zu korrigieren, kann man durch Eingeben von

```
LIST n <RETURN>
```

die Zeile mit der Nummer n auf dem Bildschirm abbilden. Mit Hilfe der Cursor-Steuertasten kann man nun den Cursor auf die fehlerhafte Stelle positionieren, den Fehler durch Ueberschreiben beseitigen und die so korrigierte Zeile durch Druecken der RETURN-Taste wieder in das Programm einfuegen.
Um eine Zeile zu loeschen, brauchen Sie nur die entsprechende Zeilennummer gefolgt von <RETURN> einzugeben. Geshiftetes <RETURN> setzt nur den Cursor an den Anfang der Folgezeile, ohne Informationen an den Interpreter zu uebergeben.
Um ein ganzes Programm aus dem Programmspeicher Ihres VC20 zu loeschen, geben Sie einfach

NEW<RETURN>

ein. Das sollten Sie auch immer dann tun, wenn Sie ein neues Programm eingeben wollen.

3.2.13 Fehlermeldungen

Jeder Fehler, den der Interpreter bei der Abarbeitung eines Programms oder einer im Direkt-Modus eingegebenen Anweisungszeile findet, fuehrt zum sofortigen Abbruch der Interpretation. Der Befehl wird nicht ausgefuehrt; es wird vielmehr eine entsprechende Fehlermeldung auf dem Bildschirm ausgegeben und der Computer geht in den Direkt-Modus zurueck. Im Anhang D finden Sie eine Zusammenstellung aller Fehlermeldungen mit ausfuehrlicher Erlaeuterung.

3.3 BASIC-Anweisungen

Im folgenden werden alle VC20-BASIC-Anweisungen detailliert beschrieben. Die Beschreibung einer jeden Anweisung sowie jeder Funktion, die in den Abschnitten 3.4.ff behandelt werden, ist nach dem folgenden Schema aufgebaut:

Format:	Zeigt das korrekte Format der Anweisung oder Funktion. Die dabei verwendete Notation ist unten beschrieben.
Zweck:	Erlaeutert den Anwendungsbereich.
Bemerkungen:	Beschreibt im Detail, wie die Anweisung oder Funktion angewendet wird.
Beispiele:	Zeigt Beispielprogramme oder -programmsegmente, die die Anwendung der Anweisung oder Funktion demonstrieren.

3.3.1 Vereinbarung der Notation

Wo immer das Format fuer eine Anweisung oder Funktion beschrieben wird, gelten folgende Schreibregeln:

* Woerter in Grossbuchstaben muessen wie angegeben geschrieben werden.

* Woerter in Gross/Kleinschrift, eingekleidet in spitze Klammern (<>), werden vom Anwender eingesetzt.

* Woerter in eckigen Klammern ([ ]) sind wahlfrei.

* Alle Sonderzeichen ausser spitzen und eckigen Klammern, also Kommas, runde Klammern, Semikolons, Dollarzeichen usw., muessen wie angegeben geschrieben werden.

* Woerter, die von ... gefolgt werden, koennen bis zur maximalen Laenge einer Befehlszeile (88 Zeichen) wiederholt werden.

3.3.2 CLOSE

Format: CLOSE ‹Filenummer›

Zweck: Beendet die Ein/Ausgabe ueber einen Ein/Ausgabe-Kanal.

Bemerkungen: ‹Filenummer› ist die Nummer zwischen 1 und 255, unter der der File (Datei) mit der OPEN-Anweisung eroeffnet wurde.
Der Zusammenhang zwischen einem bestimmten File und der Filenummer wird durch die CLOSE-Anweisung aufgehoben. Der File kann dann mit der OPEN-Anweisung unter derselben oder einer anderen Filenummer wieder eroeffnet werden oder es kann ein beliebiger anderer File unter dieser Filenummer eroeffnet werden.
CLOSE auf einen sequentiellen Ausgabefile angewendet, schreibt den letzten Datenpuffer auf den File und schliesst diesen mit einer Fileendemarke ab.

Beispiel:

```
10 OPEN 4,4
20 PRINT#4,"DIESES SIND DRUCKDATEN"
30 CLOSE 4
```

3.3.3 CLR

Format: CLR

Zweck: Setzt alle numerischen Variablen auf Null, alle Stringvariablen auf Leerstring, leert den Staelspeicher und den Speicher fuer Felder und setzt den Zeiger fuer freien Speicherplatz auf den Wert zurueck, der sich aus der Groesse des BASIC-Programms ohne alle Variablen ergibt.

Bemerkungen: CLR kann auch innerhalb eines BASIC-Programms ausgefuehrt werden. Das Programm kann dann fortgesetzt werden, wenn die oben beschriebenen Bedingungen, insbesondere solche, die sich auf GOSUB beziehen, beruecksichtigt werden.

Beispiel:

```
X=25
CLR
PRINT X
 0

READY.
```

3.3.4 CMD

Format: CMD ‹Filenummer› [,Liste von Ausdruecken]

Zweck: Adressiert ein Geraet an einer Ein/Ausgabe-Schnittstelle und laesst dieses Geraet nach der Ausgabeoperation im adressierten Zustand.

Bemerkungen: CMD hat dieselbe Parameterliste wie die PRINT#-Anweisung (s. Abschn. 3.3.24).

Beispiel:

```
REM PROGRAMMLISTE AUF DRUCKER AUSGEBEN
OPEN 4,4
CMD4,"PROGRAMMLISTE"
LIST
PRINT#4,"CMD-MODUS WIRD BEENDET"
CLOSE4
```

3.3.5 CONT

Format: CONT

Zweck: Setzt ein Programm, das durch Druecken der STOP-Taste oder die STOP- oder END-Anweisung unterbrochen bzw. beendet wurde, fort.

Bemerkungen: Die Programmausfuehrung wird unmittelbar an der Stelle, an der die Unterbrechung auftrat, fortgesetzt. Wenn die Unterbrechung nach der Textanzeige einer INPUT-Anweisung erfolgte, wird das Programm mit der Wiederholung dieser Anzeige (? oder Text) fortgesetzt.
CONT wird ueblicherweise in Verbindung mit der STOP-Anweisung zur Fehlersuche in Programmen verwendet. Nach der Programmunterbrechung koennen Zwischenergebnisse angezeigt oder durch Direkt-Modus-Anweisungen veraendert werden. Die Programmausfuehrung wird mit der Eingabe von CONT oder GOTO zusammen mit einer bestimmten Zeilennummer im Direkt-Modus fortgesetzt.
CONT ist ungueltig, wenn das Programm mit einer Fehlermeldung abgebrochen wurde oder waehrend der Unterbrechung veraendert wurde.

Beispiel: Siehe Beispiel in Abschnitt 3.3.30 (STOP).

3.3.6 DATA

Format: DATA ‹Konstantenliste›

Zweck: Speichert numerische und String-Konstanten, auf die mit der READ-Anweisung (s. Abschn. 3.3.25) zugegriffen werden kann.

Bemerkungen: DATA-Anweisungen sind nicht ausfuehrbare Anweisungen, die an beliebiger Stelle im Programm stehen koennen. Jede DATA-Anweisung kann soviele Konstanten enthalten, wie, getrennt durch Komma, in eine Befehlszeile (88 Zeichen) passen. Die Zahl der DATA-Anweisungen ist beliebig. Die READ-Anweisung liest die einzelnen DATA-Zeilen in der Reihenfolge von deren Zeilennummern. Die in diesen Zeilen enthaltenen Daten werden unabhaengig von ihrer Zahl und deren Plazierung im Programm als kontinuierliche Elementliste aufgefasst. ‹Konstantenliste› kann numerische Konstanten jeden Formats, d.h. Gleitkomma- oder ganze Zahlen enthalten. Numerische Ausdruecke sind nicht erlaubt. String-Konstanten in DATA-Anweisungen muessen nur dann in Anfuehrungsstriche (") eingekleidet werden, wenn sie Kommas, Doppelpunkte oder vor- und/oder nachlaufende signifikante Leerstellen enthalten.
Der in der READ-Anweisung deklarierte Variablentyp (numerisch oder String) muss mit dem in den zugehoerigen DATA-Anweisungen enthaltenen Konstantentyp uebereinstimmen.
Der Lesezeiger kann mit der RESTORE-Anweisung (s. Abschn. 3.3.27) auf den Anfang der ersten DATA-Anweisung gestellt werden.

Beispiel:

```
10 DATA "DAS ","WETTER ","IST ","HEUTE
","SCHOEN !"
20 FOR I=1 TO 5
30 READ A$
40 PRINT A$;
50 NEXT
RUN
DAS WETTER IST HEUTE SCHOEN !
READY.
```

3.3.7 DEF FN

Format: DEF FN<Name>[(<Parameterliste>)]=<Funktionsdefinition>

Zweck: Definiert und benennt eine vom Anwender programmierte BASIC-Funktion.

Bemerkungen: <Name> muss ein erlaubter Variablenname sein. Dieser Name, dem FN vorangestellt wird, wird als Name der Funktion betrachtet. <Parameter> ist das Argument der Funktion, das in der Funktionsdefinition durch eine Gleitkommavariable bezeichnet wird. Letztere wird dann beim Aufruf der Funktion durch den aktuellen Parameter ersetzt. <Funktionsdefinition> ist ein beliebiger Ausdruck, der die Operation, die die Funktion ausfuehren soll, beinhaltet. Die Laenge des Ausdrucks ist auf eine BASIC-Anweisungszeile (88 Zeichen) beschraenkt. In diesem Ausdruck verwendete Variablennamen dienen nur der formalen Funktionsdefinition und sind nicht mit Programmvariablen desselben Namens zu verwechseln. Ein in einer Funktionsdefinition verwendeter Variablenname kann als Parameter auftreten oder auch nicht. Ist er Parameter, so wird sein Wert beim Aufruf der Funktion ersetzt; andernfalls wird der derzeitige Wert der Variablen verwendet.
Mit DEF FN koennen keine eigenen Stringfunktionen definiert werden. Wenn im Funktionsname ein Variablentyp spezifiziert wird, so wird der Wert des Ausdruckes diesem Typ angepasst, bevor er der aufrufenden Anweisung uebergeben wird. Wenn ein Variablentyp im Funktionsnamen deklariert wurde, der nicht zu dem Typ passt, den der Ausdruck liefert, so wird eine TYPE MISMATCH-Fehlermeldung ausgegeben. Die DEF FN-Anweisung muss ausgefuehrt werden, ehe die dadurch definierte Funktion das erste mal aufgerufen wird, sonst wird eine UNDEFINED FUNCTION-Fehlermeldung ausgegeben. DEF FN kann nicht im Direkt-Modus verwendet werden.

Beispiel:

```
410 DEF FNAB(X)=X↑3/Y↑3
420 T=FNAB(I)
```

Zeile 410 definiert die Funktion, die in Zeile 420 aufgerufen wird. Dabei wird die Variable X durch den aktuellen Wert von I ersetzt. Die Variable Y behaelt den ihr zum Zeitpunkt des Funktionsaufrufes zugeordneten Wert.

3.3.8 DIM

Format: DIM‹Liste indizierter Variabler›

Zweck: Spezifiziert die Maximalwerte der Variablenindizes und legt den Speicher fuer das Variablenfeld fest.

Bemerkungen: Wenn ein Feldvariablenname ohne eine vorausgegangene DIM-Anweisung verwendet wird, so ist als maximaler Index 10 erlaubt. Wird ein groesserer Index angegeben, so wird eine BAD SUBSCRIPT-Fehlermeldung ausgegeben. Der kleinste Feldindex ist immer 0. Indizes muessen ganzzahlig sein. Die DIM-Anweisung setzt alle Elemente des spezifizierten Feldes anfaenglich auf Null bzw. Leerstring. Es koennen Matrizen mit bis zu 255 Dimensionen deklariert werden, von denen jede maximal 32767 Elemente enthalten darf. Auf jeden Fall ist die Feldgroesse durch den verfuegbaren Speicher begrenzt.

Beispiele:

```
10 DIM A(20)
20 FOR I=0 TO 20
30 READ A(I)
40 NEXT
50 DATA 1,2,3...
```

```
10 DIM R3(5,5)          :REM 36 ELEMENTE
```

```
10 DIM D$(2,2,2)        :REM 27 ELEMENTE
```

3.3.9 END

Format: END

Zweck: Beendet den Programmlauf, schliesst alle geoeffneten Files und setzt den Rechner in den Direkt-Modus.

Bemerkungen: END-Anweisungen koennen zur Programmbeendigung an jeder Stelle eines Programms stehen. Die END-Anweisung erzeugt keine Bildschirmmeldung wie z.B. BREAK bei der STOP-Anweisung. Am Ende eines Programms (letzte Zeile) ist die END-Anweisung wahlfrei. Nachdem die END-Anweisung ausgefuehrt wurde kehrt der Rechner in den Direkt-Modus zurueck.

Beispiel:

```
520 IF K>1000 THEN END
```

3.3.10 FOR...NEXT

Format: FOR <numerische Variable>=<x> TO <y> [STEP <z>]

.

.

NEXT [<numerische Variable>] [, <numerische Variable>..], wobei x, y und z numerische Ausdruecke sein muessen.

Zweck: Fuehrt eine Reihe von Anweisungen in einer Schleife in einer vorgegebenen Anzahl von Durchlaeufen aus.

Bemerkungen: <numerische Variable> wird als Zaehler fuer die Durchlaeufe verwendet. Der erste numerische Ausdruck <x> ist der Anfangswert, der zweite numerische Ausdruck <y> ist der Endwert des Zaehlers. Alle Anweisungen und Programmzeilen nach der FOR-Anweisung bis zur ersten NEXT-Anweisung werden ausgefuehrt. Dann wird der Zaehler um den Wert von <z> erhoeht und es wird geprueft, ob er groesser als der Endwert <y> geworden ist. Wenn er nicht groesser ist, verzweigt der Interpreter zurueck zu der Anweisung nach der FOR-Anweisung und der Ablauf wird wiederholt. Ist der Zaehler groesser als <y>, so wird das Programm nach der NEXT-Anweisung fortgesetzt. Dies versteht man unter einer FOR...NEXT-Schleife.
Wenn fuer <z> ein negativer Wert angegeben ist, so muss der Endwert <y> kleiner als der Anfangswert <x> sein. <y> wird in diesem Fall bei jedem Durchlauf um den Wert von <z> vermindert, bis der Zaehler kleiner als der Endwert <y> wird.
Wird STEP <z> nicht angegeben, so wird der Zaehler bei jedem Durchlauf um 1 erhoeht.
FOR...NEXT-Schleifen duerfen auch geschachtelt werden, d.h. eine Schleife darf auch innerhalb einer anderen angeordnet sein. Jeder Schleifenzaehler muss dann jedoch einen eigenen Namen erhalten. Fuer alle Zaehler in geschachtelten Schleifen reicht eine NEXT-Anweisung, gefolgt von den einzelnen Zaehlervariablen in der richtigen Reihenfolge und durch Kommas getrennt, wenn die einzelnen NEXT-Anweisungen unmittelbar aufeinander folgen wuerden.
Die Variablen in der NEXT-Anweisung koennen weggelassen werden. In diesem Fall bezieht sich jede NEXT-Anweisung auf die zuletzt interpretierte FOR-Anweisung. Findet der Interpreter eine NEXT-Anweisung ohne vorangegangene FOR-Anweisung, so gibt er eine NEXT WITHOUT FOR-Fehlermeldung aus und bricht das Programm ab.
Wegen des begrenzten Stapelspeichers duerfen nur maximal 9 FOR...NEXT-Schleifen ineinander geschachtelt werden.

Beispiel 1:

```
10 REM GESCHACHTELTE SCHLEIFEN
20 FOR I=1 TO 3
30 FOR J=1 TO 3
40 PRINT I;J
50 NEXT J,I
RUN
 1  1
 1  2
 1  3
 2  1
 2  2
 2  3
 3  1
 3  2
 3  3

READY.
```

Beispiel 2:

```
10 REM VARIABLENAENDERUNG NACH SETZEN DER
SCHLEIFE
20 K=10
30 FOR I=1 TO K STEP 2
40 PRINT I;
50 K=K+10
60 PRINT K
70 NEXT
RUN
 1  20
 3  30
 5  40
 7  50
 9  60

READY.
```

Beispiel 3:

```
10 REM DER ZWEITE WERT IST KLEINER ALS DER ERSTE
20 J=0
30 FOR I=1 TO J
40 PRINT I
50 NEXT
RUN
 1

READY.
```

In diesem Beispiel wird die Schleife nur einmal durchlaufen, weil der Anfangswert groesser als der Endwert ist, was jedoch erst beim Erreichen der NEXT-Anweisung geprueft wird.

Beispiel 4:

```
10 REM EINE SCHON VORHER BENUTZTE VARIABLE ALS
ZAEHLER
20 I=5
30 FOR I=1 TO I+5
40 PRINT I;
50 NEXT
RUN
 1  2  3  4  5  6
READY.
```

Hier wird die Schleife sechsmal durchlaufen, da der Schleifenzaehler I nach dem Setzen in Zeile 10 noch einmal in Zeile 20 gesetzt wird.

Beispiel 5:

```
10 REM INTEGER-VARIABLE ALS ZAEHLER
20 FOR I%=1 TO 10
30 PRINT I%
40 NEXT
RUN
?SYNTAX ERROR IN 10
READY.
```

Achtung: Integer-Variablen als Schleifenzaehler sind verboten!

3.3.11 GET und GET#

Format: GET[#<logische Filenummer>,]<Variable>

Zweck: Liest ein Zeichen aus einem File (Datei) und weist dieses Zeichen einer Variablen zu.

Bemerkungen: GET ohne die Angabe eine logischen Filenummer liest aus dem Tastaturpuffer den Code der zuletzt gedrueckten Taste und weist ihn der spezifizierten Variablen (numerisch oder String) zu. Wurde keine Taste gedrueckt, so liefert GET den Wert 0 bzw. einen Leerstring.
GET# liest ein Zeichen aus dem unter der logischen Filenummer eroeffneten File. Wurde ein logischer File mit der Geraetenummer 0 eroeffnet, so ist GET# identisch mit GET, da der Tastatur die Geraetenummer 0 zugeordnet ist.

Beispiel:

```
10 PRINT"WARTEN AUF GEDRUECKTE TASTE"
20 GET A$:IF A$="" THEN 20
```

3.3.12 GOSUB...RETURN

Format:

```
GOSUB<Zeilennummer>
.
.
.
RETURN
```

Zweck: Verzweigt in ein Unterprogramm, das mit <Zeilennummer> beginnt und kehrt nach Ausfuehrung des Unterprogramms ins Hauptprogramm zurueck

Bemerkungen: <Zeilennummer> ist die erste Zeile des Unterprogramms. Die RETURN-Anweisung(en) in einem Unterprogramm bewirken einen Ruecksprung zu der Anweisung, die der zuletzt interpretierten GOSUB-Anweisung folgt. Ein Unterprogramm kann mehrere RETURN-Anweisungen enthalten, wenn ein Ruecksprung von logisch unterschiedlichen Stellen erforderlich ist. Ein Unterprogramm kann an beliebigen Stellen im Hauptprogramm stehen, muss jedoch von diesem unterschieden werden. Um unbeabsichtigtes Durchlaufen eines Unterprogramms zu vermeiden, kann vor dem Unterprogramm eine STOP-, END- oder GOTO-Anweisung stehen, die die Programmsteuerung um das Unterprogramm herumfuehrt. Wenn die Unterprogramme am Anfang des Hauptprogramms stehen, werden diese schneller ausgefuehrt. Unterprogramme koennen in bis zu 23 Ebenen geschachtelt werden.

Beispiel:

```
10 GOSUB 40
20 PRINT"AUS DEM UNTERPROGRAMM ZURUECK"
30 END
40 PRINT"IM UNTERPROGRAMM"
50 RETURN
RUN
IM UNTERPROGRAMM
AUS DEM UNTERPROGRAMM ZURUECK
READY.
```

3.3.13 GOTO

Format: GOTO <Zeilennummer>

Zweck: Springt unbedingt aus der normalen Programmabfolge zu einer spezifizierten Zeilennummer.

Bemerkungen: Wenn <Zeilennummer> eine Zeile mit einer ausfuehrbaren Anweisung kennzeichnet, werden diese und die darauf folgenden Zeilen abgearbeitet. Existiert die spezifizierte Zeile nicht, so wird das Programm mit der naechsten auf die spezifizierte Zeile folgenden Zeile fortgesetzt.

Beispiel:

```
10 READ R
20 PRINT"R = ";R,
30 A=3.14*R↑2
40 PRINT"FLAECHE =";A
50 GOTO 10
60 DATA 5,7,12
RUN
 R = 5        FLAECHE = 78.5
 R = 7        FLAECHE = 153.86
 R = 12       FLAECHE = 452.16
?OUT OF DATA ERROR IN 10
READY.
```

3.3.14 IF...THEN und IF...GOTO

Format: IF <Ausdruck> THEN <Anweisung(en)><Zeilennummer>

Format: IF <Ausdruck> GOTO <Zeilennummer>

Zweck: Es wird hinsichtlich des Programmablaufes eine Entscheidung, basierend auf dem Ergebnis eines Ausdrucks, gefaellt.

Bemerkungen: Wenn das Ergebnis von <Ausdruck> von Null verschieden ist, wird die THEN- oder GOTO-Klausel ausgefuehrt. THEN kann entweder von einer Zeilennummer zum Verzweigen oder von einer oder mehrere Anweisungen gefolgt werden. GOTO wird immer von einer Zeilennummer gefolgt. Ist das Ergebnis von <Ausdruck> Null, so wird die THEN- oder GOTO-Klausel ignoriert und das Programm mit der folgenden Befehlszeile fortgesetzt.
IF...THEN-Anweisungen koennen auch geschachtelt werden, wobei die Schachtelungen nur durch die Befehlszeilenlaenge (88 Zeichen) begrenzt werden:

```
IF A=B THEN IF B=C THEN PRINT"A=C"
```

Wird eine im Direkt-Modus eingegebene IF...THEN-Anweisung von einer Zeilennummer gefolgt, so generiert der Interpreter eine UNDEFINED LINE-Fehlermeldung, selbst wenn vorher eine Zeile mit dieser Nummer eingegeben wurde.
Bei Verwendung von IF zum Testen eines Wertes, der sich aus einer Gleitkommaberechnung ergeben hat, ist zu beachten, dass die interne Darstellung des Wertes ungenau sein kann. Deshalb sollte ein Test immer fuer den Bereich gemacht werden, innerhalb dessen die Genauigkeit variiert. Es sollte also z.B. beim Testen einer Variablen auf den berechneten Wert 1.0 folgendermassen verfahren werden:

```
IF ABS(A-1.0)<=1.0E-6 THEN ...
```

Dieser Test liefert das Ergebnis "wahr", wenn der Wert von A gleich 1.0 mit einem relativen Fehler von weniger als 1.06E-6 ist.

Beispiel 1:

```
100 IF I THEN GET I
```

Diese Anweisung prueft auf eine gedrueckte Taste, falls der Wert von I nicht Null ist.

Beispiel 2:

```
100 IF (I>10) AND (I<20) THEN DB=1979-1:GOTO 300
110 PRINT"BEREICHSUEBERSCHREITUNG"
```

Wenn I groesser als 10 und kleiner als 20 ist, wird DB berechnet und das Programm wird mit Zeile 300 fortgesetzt. Sonst wird Zeile 110 ausgefuehrt.

3.3.15 INPUT

Format: INPUT [“<Anzeigestring>”;]<Variablenliste>

Zweck: Erlaubt Dateneingabe ueber die Tastatur waehrend der Programmausfuehrung.

Bemerkungen: Trifft der Interpreter auf eine INPUT-Anweisung, so wird der Programmlauf angehalten, ein Fragezeichen wird auf dem Bildschirm ausgegeben, um zu zeigen, dass das Programm Dateineingabe erwartet und der Cursor blinkt. Wurde <Anzeigestring> spezifiziert, so wird dieser vor dem Fragezeichen abgebildet. Die geforderten Daten koennen dann ueber die Tastatur eingegeben werden. Diese Daten werden der (den) Variablen in der <Variablenliste> zugeordnet; deshalb muss die Zahl der Dateneinheiten (getrennt durch Kommas) mit der Zahl der Variablen in der Liste uebereinstimmen.
Die Variablen in der Liste duerfen Namen fuer numerische und String-Variablen (auch Feldvariablen) sein. Der Typ jeder eingegebenen Dateneinheit muss mit dem Typ der korrespondierenden Variablen uebereinstimmen. Eingegebene Strings muessen nicht in Anfuehrungsstriche (") eingekleidet werden, es sei denn, sie enthalten Kommas und/oder Doppelpunkte. Die Dateneingabe ist auf die Laenge einer logischen Bildschirmzeile (88 Zeichen) begrenzt. Wegen des Fragezeichens koennen also hoechstens 86 Zeichen eingegeben werden. Wird diese Zahl ueberschritten, nimmt INPUT die zuletzt eingegebene logische Bildschirmzeile als gesamte Eingabe. Als logische Bildschirmzeile werden Bildschirmdaten von bis zu 88 Zeichen, gerechnet vom Zeilenanfang bis zum Wagenruecklauf-Code (RETURN-Taste), betrachtet.
Wird bei INPUT der falsche Datentyp eingegeben, also z.B. Stringdaten anstatt numerischer, so wird die Meldung ?REDO FROM START ausgegeben und es wird auf die richtige Eingabe gewartet.
Im Direkt-Modus ist INPUT nicht erlaubt. In diesem Fall wird die ILLEGAL DIRECT-Fehlermeldung ausgegeben. Werden bei INPUT zuviele Dateneinheiten eingegeben, wird die EXTRA IGNORED-Meldung ausgegeben. Bei zuwenigen Dateneinheiten werden die fehlenden mit ?? nachgefordert.

Beispiel 1:

```
10 REM EINGABE VON ZUVIELEN ZEICHEN
20 INPUT A$
30 PRINT:PRINT A$
RUN
? DDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDD
****<RETURN>

****
```

Es wurde 86 mal D und 4 mal * eingegeben. Da INPUT bei Laengenueberschreitung nur die letzte logische Bildschirmzeile beruecksichtigt, werden nur die 4 * der Stringvariablen A$ zugeordnet.

Beispiel 2:

```
10 INPUT X
20 PRINT X"ZUM QUADRAT IST"X 2
RUN
? 5<RETURN>
 5 ZUM QUADRAT IST 25
READY.
```

Beispiel 3:

```
10 PI=3.14159265
20 INPUT"RADIUS EINGEBEN";R
30 A=PI*R↑2
40 PRINT"KREISFLAECHE IST";A
50 PRINT
60 GOTO 20
RUN
RADIUS EINGEBEN? 7.4<RETURN>
KREISFLAECHE IST 172.033614

RADIUS EINGEBEN? usw.
```

3.3.16 INPUT#

Format: INPUT#<logische Filenummer>,<Variablenliste>

Zweck: Liest Daten aus einem sequentiellen oder Direktzugriffs-File und weist sie Programmvariablen zu.

Bemerkungen: <logische Filenummer> ist die Nummer, unter der der File als Eingabefile mit der OPEN-Anweisung (s. Abschnitt 3.3.22) eroeffnet wurde. <Variablenliste> enthaelt die Namen der Variablen, denen die Datenelemente aus dem File zugewiesen werden. Die Datentypen muessen den Variablentypen entsprechen. INPUT# gibt kein ? als Anzeige aus, wenn als Eingabegeraet die Tastatur gewaehlt wurde.
Die Dateneinheiten in dem File muessen genauso angeordnet sein, als wuerden sie ueber die Tastatur eingegeben. Vor- und nachlaufende Leerstellen, Wagenruecklauf- und Zeilenvorschub-Codes werden ignoriert. Trennzeichen zwischen Variableninhalten koennen Komma oder Doppelpunkt sein; der Wagenruecklaufcode trennt auf jeden Fall einzelne Datenelemente voneinander. Bei Nichtuebereinstimmung von Daten- und Variablentyp wird eine FILE DATA ERROR-Fehlermeldung ausgegeben. Beim Versuch, Daten am Fileende zu lesen oder bei Zeitueberschreitung an der Eingabeschnittstelle (s. Abschn. 3.4.21) liefert INPUT# einen Wagenruecklauf-Code (CHR$(13)). INPUT# kann bei Strings maximal 88 Zeichen lesen.

Beispiel:

```
10 REM LESEN VON KASSETTE BIS ZUM FILEENDE
20 OPEN 3,1,0
30 INPUT#3,A$
40 IF STATUS AND 64 THEN FE=1
50 PRINT A$
60 IF FE THEN CLOSE 3:END
70 GOTO 30
```

Die Abfrage des Rechner-Statuswortes (hier in Zeile 40) wird in Abschnitt 3.4.21 ausfuehrlich beschrieben.

3.3.17 LET

Format: [LET]<Variable>=<Ausdruck>

Zweck: Weist den Wert eines Ausdrucks einer Variablen zu.

Bemerkungen: Das Wort LET ist wahlfrei, d.h. bei der Zuweisung eines Wertes zu einer Variablen genuegt das Gleichheitszeichen.

Beispiel:

```
110 LET D=12:LET E=12*2
120 LET F=144/12
130 LET SUM=D+E+F
```

gleichbedeutend mit:

```
110 D=12:E=12*2
120 F=144/12
130 SUM=D+E+F
```

3.3.18 LIST

Format 1: LIST [<Zeilennummer>]

Format 2: LIST [<Zeilennummer>]-[<Zeilennummer>]

Zweck: Listet einen Teil oder ein ganzes Programm auf das gegenwaertig aktivierte Ausgabegeraet (Bildschirm, Drucker, Kassette, Floppy Disk).

Bemerkungen: Der Rechner wird nach der Ausfuehrung von LIST immer in den Direkt-Modus gesetzt.

Format 1: Wenn <Zeilennummer> weggelassen wird, wird das ganze Programm beginnend mit der kleinsten Zeilennummer gelistet. Das Listen wird entweder durch das Programmende oder durch Druecken der STOP-Taste beendet. Wird <Zeilennummer> angegeben, so wird nur diese eine Zeile gelistet.

Format 2: Dieses Format erlaubt folgende Moeglichkeiten:

1. Wird nur die erste Zeilennummer angegeben, so wird das Programm ab dieser Zeilennummer gelistet.

2. Wird nur die zweite Zeilennummer angegeben, so wird das Programm vom Anfang bis einschliesslich dieser Zeile gelistet.

3. Sind beide Zeilennummern angegeben, wobei die zweite groesser als die erste sein muss, so wird nur dieser Programmbereich gelistet.

Beispiele:

LIST	Listet das gesamte im Speicher befindliche Programm.
LIST 500	Listet Zeile 500.
LIST 150-	Listet alle Zeilen von Zeile 150 einschliesslich bis Programmende.
LIST -1000	Listet alle Zeilen vom Anfang des Programms bis Zeile 1000 einschliesslich.
LIST 150-1000	Listet alle Zeilen von Zeile 150 bis Zeile 1000 einschliesslich.

3.3.19 LOAD

Format: LOAD "<Filename>"[,<Geraetenummer>]

Zweck: Laedt einen Programmfile von einem externen Speichergeraet (Kassette, Floppy Disk) in den Speicher des Rechners.

Bemerkungen: <Filename> ist der Name, unter dem das Programm mit der SAVE-Anweisung (s. Abschn. 3.3.30) auf ein externes Speichergeraet gespeichert wurde. Wird <Geraetenummer> weggelassen, so wird das Programm von Geraet Nr. 1 (Kassettenstation) geladen. Durch LOAD werden alle eroeffneten Files geschlossen sowie alle gesetzten Variablen und das ggfs. im Speicher befindliche Programm geloescht, ehe das spezifizierte Programm geladen wird.
Wenn LOAD innerhalb eines Programms ausgefuehrt wird, wird das dadurch geladene Programm sofort gestartet, wobei alle eroeffneten Files offen bleiben. Dadurch koennen mehrere Programme oder Programmsegmente miteinander verkettet werden. Gesetzte Variablen bleiben durch die Verkettung erhalten. Das nachgeladene Programm darf jedoch hoechstens genau so gross sein, wie das aufrufende.

Beispiel: Laden eines Programms von Kassette:

```
LOAD "TESTPRG"
```

3.3.20 NEW

Format:	NEW
Zweck:	Loescht das gegenwaertig im Speicher befindliche Programm sowie alle Variablen.
Bemerkungen:	NEW wird gewoehnlich im Direkt-Modus eingegeben, ehe ein neues Programm editiert wird. Der Interpreter setzt den Rechner auf jeden Fall nach Ausfuehren von NEW in den Direkt-Modus.

3.3.21 ON...GOSUB und ON...GOTO

Format:

ON <Ausdruck> GOTO <Liste mit Zeilennummern>

ON <Ausdruck> GOSUB <Liste mit Zeilennummern>

Zweck:

Verzweigt zu einer von mehreren spezifizierten Zeilennummern in Abhaengigkeit vom Wert, der von <Ausdruck> geliefert wird.

Bemerkungen:

Der Wert von <Ausdruck> bestimmt, zu welcher Zeilennummer aus der Liste das Programm verzweigt. Wenn der Wert z.B. 3 ist, so stellt die 3. Zeilennummer in der Liste das Sprungziel dar. Vor der Verzweigung wird der Wert auf jeden Fall in eine ganze Zahl umgewandelt, d.h., ggf. vorhandene Dezimalstellen werden abgeschnitten.
Bei der ON...GOSUB-Anweisung muss jede spezifizierte Zeilennummer die Anfangszeile eines Unterprogramms kennzeichnen.
Ist der Wert des Ausdrucks negativ, so wird eine ILLEGAL QUANTITY-Fehlermeldung ausgegeben. Ist er Null oder groesser als die Anzahl der in der Liste angegebenen Zeilennummern, so wird das Programm mit der auf diese Anweisung folgenden Zeile fortgesetzt.

Beispiel:

```
100 ON L-1 GOTO 150,300,320,220
```

3.3.22 OPEN

Format: OPEN <logische Filenummer>[,<Geraetenummer>
[,<Sekundaeradresse>[,"<Filename>"]]]

Zweck: Eroeffnet einen Ein/Ausgabe-Kanal ueber die serielle, die RS-232-, die IEEE- oder die interne Schnittstelle.

Bemerkungen: Die logische Filenummer muss zwischen 1 und 255 liegen. Falls keine Geraetenummer angegeben wird, wird 1 angenommen (Kassettenstation). Fuer Sekundaeradressen und Filenamen existieren keine Voreinstellungen. Mit jeder GET#-, INPUT#- oder Print#-Anweisung werden die Geraetenummer und eine ggfs. spezifizierte Sekundaeradresse ueber die Schnittstelle gesendet.
Bei Floppy-Disk-Files wird der Filetyp mit P (Programmfile) angenommen, falls nicht S (sequentieller Datenfile) getrennt durch Komma an den Filenamen angefuegt wurde. Wird nach einem weiteren Komma ein W angegeben, so wird der spezifizierte File zum Schreiben eroeffnet, andernfalls zum Lesen.
Bei Kassettenfiles bezeichnet eine Sekundaeradresse 0 einen Lesefile, 1 einen Schreibfile und 2 einen Schreibfile mit anschliessender Bandendemarke.
Files koennen fuer Tastatur (Geraetenr. 0), Kassettengeraet (Geraetenr. 1), RS-232-Schnittstelle (Geraetenr. 2, s. Anhang A) oder Bildschirm (Geraetenr. 3) eroeffnet werden. Geraetenummern groesser als 3 beziehen sich auf Geraete, die an der IEEE-Schnittstelle angeschlossen werden koennen (z.B. 4 fuer Drucker, 8 fuer Floppy Disk).

Beispiel:

```
10 REM EROEFFNEN EINES KASSETTEN-SCHREIBFILES MIT
20 REM ANSCHLIESSENDER BANDENDEMARKE
30 OPEN 6,1,2,"DATENFILE"
40 FOR I=1 TO 10
50 PRINT#6,CHR$(I)
60 NEXT
70 CLOSE 6
```

3.3.23 POKE

Format: POKE I,J

I und J muessen ganzzahlige Ausdruecke sein.

Zweck: Schreibt eine 8-Bit-Binaerinformation in eine spezifizierte Speicherzelle.

Bemerkungen: Der Wert des Integer-Ausdrucks I bezeichnet die zu beschreibende Speicherzelle; der Wert des Integerausdrucks J bezeichnet das zu speichernde Datum. I muss im Bereich zwischen 0 und 65535 und J im Bereich zwischen 0 und 255 liegen.
Das Gegenstueck zur POKE-Anweisung ist die PEEK-Funktion (s. Abschn. 3.4.13), deren Argument eine Speicherzelle bezeichnet, deren Inhalt ausgelesen werden soll.
POKE und PEEK sind effektive Hilfsmittel fuer die Datenspeicherung, das Laden von Unterprogrammen in Machinensprache sowie die Uebergabe von Parametern und Ergebnissen zwischen BASIC-Hauptprogrammen und Maschinensprache-Unterprogrammen.

Beispiel 1:

```
10 REM FARB- UND ZEICHENSTEURUNG
20 POKE 7680,1:POKE 38400,6
```

Hier wird der Buchstabe A in blauer Farbe in der HOME-Position des Bildschirms abgebildet (s.a. Abschn. 2.6).

Beispiel 2:

```
10 REM ALLE TASTEN HABEN WIEDERHOLFUNKTION
20 POKE 651,128
30 REM NUR DIE CURSORSTEUERTASTEN HABEN
40 REM WIEDERHOLFUNKTION
50 POKE 651,127
```

3.3.24 PRINT und PRINT#

Format: PRINT[#<log. Filenummer>,][<Liste von Ausdruecken>]

Zweck: Gibt Daten an den Bildschirm oder ueber einen spezifizierten Ausgabekanal aus.

Bemerkungen: Wird <Liste von Ausdruecken> nicht angegeben, so wird eine Leerzeile gedruckt. Andernfalls werden die Ausdruecke ermittelt und deren Werte an das in der zugehoerigen OPEN-Anweisung unter der logischen Filenummer spezifizierte Ausgabegeraet ausgegeben. Es sind numerischen und/oder Stringausdruecke erlaubt. Stringkonstanten muessen in Anfuehrungsstriche eingekleidet werden.
Die Position jeder zu druckenden Dateneinheit wird durch die Interpunktion, die die Dateneinheiten in der Liste voneinander trennt, bestimmt. Der BASIC-Interpreter teilt die Druckzeile in Druckzonen von je 10 Leerstellen ein. Ein Komma in der Liste von Ausdruecken bewirkt, dass der Wert des darauffolgenden Ausdrucks ab dem Anfang der naechsten Druckzone gedruckt wird, wohingegen ein Semikolon bewirkt, dass der naechste Wert unmittelbar hinter den vorausgehenden Wert gedruckt wird. Eine oder mehrere Leerstellen zwischen den Ausdruecken haben dieselbe Wirkung wie Semikolon. Ein Komma oder Semikolon am Ende einer Liste von Ausdruecken bedeutet, dass die Werte der naechsten PRINT-Anweisung in der naechsten Druckzone derselben Zeile oder unmittelbar anschliessend in derselben Zeile gedruckt werden. Bei beiden Zeichen am Ende einer Liste von Ausdruecken wird ein Wagenruecklauf-Code unterdrueckt. Ist die zu druckende Zeile laenger als 22 Zeichen, so wird das Drucken in der naechsten physikalischen Zeile fortgesetzt.
Gedruckten Zahlenwerten folgt immer eine Leerstelle. Positiven Zahlenwerten ist eine Leerstelle, negativen ein Minus-Zeichen vorangestellt. Jede Zahl zwischen 0 und 0.01 wird in der wissenschaftlichen Exponentialdarstellung (s. Abschn. 2.4.3) wiedergegeben. Die PRINT-Anweisung (**nicht PRINT#**) kann durch ? abgekuerzt werden.

Beispiel 1:

```
10 X=5
20 PRINT X+5,X-5,X*(-5),X↑5
RUN
 10           0           -25          3125
READY.
```

Die Kommas zwischen den Ausdruecken in Zeile 20 bewirken, dass jeder Wert an den Anfang einer 10 Leerstellen breiten Druckzone gedruckt wird.

Beispiel 2:

```
10 INPUT X
20 PRINT X"ZUM QUADRAT IST"X↑2"UND";
30 PRINT X"ZUM KUBIK IST"X↑3
40 PRINT
50 GOTO 10
RUN
? 9RETURN
 9 ZUM QUADRAT IST 81 UND 9 ZUM KUBIK IST 729

?  usw.
```

Hier bewirkt das Semikolon am Ende von Zeile 20, dass die Werte beider PRINT-Anweisungen in den Zeilen 20 und 30 in dieselbe Zeile gedruckt werden. Zeile 40 bewirkt das Drucken einer Leerzeile.

Beispiel 3:

```
10 FOR X=1 TO 5
20 J=J+5
30 K=K+10
40 ?J;K;
50 NEXT
RUN
 5  10  10  20  15  30  20  40  25  50
READY.
```

Bei diesem Beispiel wurde in Zeile 40 fuer die PRINT-Anweisung das ?-Zeichen gewaehlt. Die Semikolons bewirken das Drucken der einzelnen Werte unmittelbar hintereinander getrennt durch 2 Leerstellen (jede Zahl wird von einer Leerstelle gefolgt, positiven Zahlen ist eine Leerstelle vorangestellt). Wird das Programm mit LIST ausgelistet, so wird das ?-Zeichen in Zeile 40 durch das Wort PRINT ersetzt.

3.3.25 READ

Format: READ<Variablenliste>

Zweck: Liest Daten aus einer DATA-Anweisung und weist sie Variablen zu (s.a. Abschn. 3.3.6).

Bemerkungen: Eine READ-Anweisung darf nur in Verbindung mit einer DATA-Anweisung benutzt werden. Jeder Variablen aus der Liste, die eine numerische oder eine String-Variable sein kann, wird immer nur ein Wert aus der DATA-Anweisung zugewiesen. Daten- und Variablentypen muessen uebereinstimmen. Andernfalls wird eine SYNTAX ERROR-Fehlermeldung ausgegeben.
Eine einzelne READ-Anweisung kann sequentiell auf mehrere DATA-Anweisungen zugreifen wie auch mehrere READ-Anweisungen auf eine DATA-Anweisung zugreifen koennen. Wenn die Anzahl von Variablen in der Variablenliste groesser ist als die Anzahl von Elementen in der (den) DATA-Anweisung(en), wird eine OUT OF DATA-Fehlermeldung ausgegeben. Sind weniger Variablen in der Liste spezifiziert als Elemente in der (den) DATA-Anweisung(en) vorhanden sind, so lesen folgende READ-Anweisungen die noch nicht gelesenen Elemente. Folgen in einem solchen Fall keine weiteren READ-Anweisungen, so bleiben ueberzaehlige Datenelemente unberuecksichtigt. Um DATA-Anweisungen wiederholt von Anfang an zu lesen, kann die RESTORE-Anweisung (s. Abschn. 3.3.27) verwendet werden.

Beispiel 1:

```
80 FOR I=1 TO 10
90 READ A(I)
100 NEXT
110 DATA 3.08,5.19,3.12,3.98,4.24
120 DATA 5.08,5.55,4.00,3.16,3.37
.
```

In diesem Programmsegment werden die Elemente der DATA-Anweisungen der Zeilen 110 und 120 in das Feld A gelesen.

Beispiel 2:

```
10 PRINT "PLZ","STADT","LAND"
20 READ PZ,S$,L$
30 DATA 6000,"FRANKFURT","HESSEN"
40 PRINT PZ,S$,L$
RUN
PLZ          STADT       LAND
 6000        FRANKFURT   HESSEN
READY.
```

Hier werden numerische und String-Daten aus der DATA-Anweisung in Zeile 30 gelesen und ausgedruckt.

3.3.26 REM

Format: REM [<Kommentar>]

Zweck: Fuegt erlaeuternden Kommentar in ein Programm ein.

Bemerkungen: REM-Anweisungen werden nicht ausgefuehrt, jedoch exakt wiedergegeben, wenn das Programm gelistet wird.
Von einer GOTO- oder GOSUB-Anweisung kann zu einer REM-Anweisung verzweigt werden. Das Programm wird dann mit der naechsten auf die REM-Anweisung folgenden ausfuehrbaren Anweisung fortgesetzt.

Beispiel:

```
10 REM BERECHNUNG DER MITTLEREN GESCHWINDIGKEIT
20 FOR I=1 TO 20
30 SUM=SUM+V(I)
40 NEXT
50 VM=SUM/(I-1)
```

3.3.27 RESTORE

Format: RESTORE

Zweck: Setzt den Lesezeiger der READ-Anweisung auf den Anfang der ersten DATA-Anweisung im Programm.

Bemerkungen: Nach der Ausfuehrung einer RESTORE-Anweisung greift die naechste READ-Anweisung auf das erste Datenelement in der ersten DATA-Anweisung im Programm zu.

Beispiel:

```
10 READ A,B,C
20 PRINT A,B,C
30 RESTORE
40 READ D,E,F
50 PRINT D,E,F
60 DATA 57,68,79
RUN
 57          68          79
 57          68          79
READY.
```

3.3.28 RUN

Format: RUN [<Zeilennummer>]

Zweck: Startet das gegenwaertig im Programmspeicher befindli- che BASIC-Programm.

Bemerkungen: Wird <Zeilennummer> spezifiziert, so wird das Programm mit der dadurch bezeichneten Zeile gestartet. Andernfalls beginnt die Ausfuehrung mit der niedrigsten Zeilennummer. Vor dem Programmstart wird durch RUN zuerst CLR (s. Abschn. 3.3.3) ausgefuehrt.
Ein Programm wird durch Ruecksetzen des Rechners in den Direkt-Modus beendet, wenn:

1. keine ausfuehrbaren Zeilen mehr vorhanden sind.

2. eine END- oder STOP-Anweisung ausgefuehrt wurde.

3. ein Fehler waehrend der Ausfuehrung auftritt.

3.3.29 SAVE

Format: SAVE [“<Filename>“[,<Geraetenummer>[,<Befehl>]]]

Zweck: Speichern eines BASIC-Programmfiles auf einem spezifizierten Ausgabegeraet.

Bemerkungen: Wenn keine Geraetenummer angegeben wird, so wird das im Programmspeicher befindliche BASIC-Programm auf Kassette (Geraet Nr. 1) gespeichert. Als Befehl kann bei Speicherung auf Kassette eine Null (keine Bandendemarke nach dem Programmfile) oder ein von Null verschiedener Wert (Bandendemarke nach dem Programmfile) angegeben werden. Wird <Filename> nicht vergeben (nur bei Kassettenspeicherung erlaubt), so wird ueberhaupt kein Name gespeichert. Wird als Filename eine Stringvariable gesetzt, so muss diese in Klammern angegeben werden.

Beispiele:

```
SAVE

SAVE "TESTPRG"

SAVE(A$),1,0
```

3.3.30 STOP

Format: STOP

Zweck: Bricht ein laufendes Programm ab und setzt den Rechner in den Direkt-Modus.

Bemerkungen: STOP-Anweisungen duerfen an beliebiger Stelle in einem Programm stehen. Wird ein STOP ausgefuehrt, so meldet der Interpreter dies mit:

BREAK IN nnnnn (nnnnn = Zeilennummer)

Die STOP-Anweisung schliesst nicht, wie die END-Anweisung ggf. eroeffnete Dateien. Nach einer STOP-Anweisung kann das Programm durch Eingabe von CONT im Direkt-Modus fortgesetzt werden (s.a. Abschn. 3.3.5).

Beispiel:

```
10 INPUT A,B,C
20 K=(A+3)/2:L=B*3
30 STOP
40 M=C*K+100:PRINT M
RUN
? 1,2,3<RETURN>
BREAK IN 30
READY.
CONT<RETURN>
 106
READY.
```

3.3.31 SYS

Format: SYS <Ausdruck>[(<Parameterliste>)]

Zweck: Uebergibt die Programmsteuerung an ein Unterprogramm in Maschinensprache, das bei einer spezifizierten Adresse beginnt (s.a. Abschn. 3.4.27, USR-Funktion).

Bemerkungen: Der Wert von Ausdruck muss eine ganze Zahl zwischen 0 und 65535 sein. Er bezeichnet die Adresse im Programmspeicher des VC20, bei der das Unterprogramm in Maschinensprache beginnt. Die Rueckkehr in das BASIC-Hauptprogramm erfolgt durch den Assemblerbefehl RTS.
In <Parameterliste> koennen Parameter angegeben werden, die dem Maschinensprache-Unterprogramm uebergeben werden sollen.

Beispiel: SYS 7*2↑12(X,Y)

3.3.32 VERIFY

Format: VERIFY [“<Filename>“[,<Geraetenummer>]]

Zweck: Vergleicht ein gegenwaertig im Programmspeicher befindliches Programm mit einem auf einem spezifizierten Ausgabegeraet gespeicherten Programm und meldet ggfs. Unterschiede.

Bemerkungen: Die Geraetenummer ist mit 1 (Kassettenstation) voreingestellt. Wird kein Filename angegeben (nur bei Kassettenfiles erlaubt), so erfolgt der Vergleich mit dem ersten auf Kassette gefundenen Programm.

Besipiel:

```
VERIFY "PRGFILE"
PRESS PLAY ON TAPE
OK
FOUND PRGFILE
VERIFYING
VERFY OK oder VERIFYING ERROR
READY.
```

3.3.33 WAIT

Format: WAIT <Adresse>,I[,J]

Zweck: Haelt die Programmausfuehrung an, bis eine angegebene Speicherzelle des VC20 ein spezifiziertes Bitmuster angenommen hat.

Bemerkungen: Zur Pruefung des spezifizierten Bitmusters wird zwischen dem Inhalt der durch <Adresse> gekennzeichneten Speicherzelle und dem Integer-Ausdruck J eine exklusive ODER-Verknuepfung gebildet. Dieses Ergebnis wird durch ein logisches UND mit dem Integer-Ausdruck I verknuepft. Wenn das Ergebnis Null ist, wird das Bitmuster der spezifizierten Speicherzelle erneut getestet. Erst wenn das Ergebnis von Null verschieden ist, wird die naechste BASIC-Anweisung ausgefuehrt. Wenn J weggelassen wird, wird sein Wert mit Null angenommen.

Achtung: Die WAIT-Anweisung kann nicht mit der STOP-Taste abgebrochen werden.

Beispiele:

```
WAIT 37157,64,64
```

Die Programmausfuehrung wird solange angehalten, bis eine Taste der Kassettenstation gedrueckt wird.

```
WAIT A,2↑N      mit N=0,1,....,7
```

Die Programmausfuehrung wird solange angehalten bis das Bit N der durch A spezifizierten Speicherzelle logisch 1 ist.

3.4 BASIC-Funktionen

In den folgenden Abschnitten werden die bereits erwaehnten Funktionen, die der VC20-BASIC-Interpreter zur Verfuegung stellt, detailliert beschrieben. Jede Funktion kann einfach ohne weitere Definition in einem Programm aufgerufen werden. Die Argumente der Funktionen werden immer in runde Klammern eingeschlossen angegeben. In den in den naechsten Abschnitten beschriebenen Funktionen werden die Argumente wie folgt abgekuerzt:

X und Y	beliebige numerische Ausdruecke
I und J	beliebige Integer-Ausdruecke
X$ und Y$	beliebige String-Ausdruecke

Wird bei einer Funktion, die einen Integerausdruck als Argument verlangt, ein Gleitkommaausdruck angegeben, so wird dessen Wert vorher durch Abschneiden der Dezimalstellen in einen Integer-Wert umgewandelt. Im uebrigen gilt die in Abschnitt 3.3.1 vereinbarte Notation.

3.4.1 ABS

Format: ABS(X)

Wirkung: Liefert den Absolutwert des Ausdrucks X.

Beispiel:

```
PRINT ABS(7*(-5))
 35
READY.
```

3.4.2 ASC

Format: ASC(X$)

Wirkung: Liefert einen numerischen, ganzzahligen Wert zwischen 0 und 255, der den ASCII-Code des ersten Zeichens des Strings X$ repraesentiert. Ist X$ ein Leerstring, so wird eine ILLEGAL QUANTITY-Fehlermeldung ausgegeben.

Beispiel:

```
10 A$="TEST"
20 PRINT ASC(A$)
RUN
 84
READY.
```

3.4.3 ATN

Format: ATN(X)

Wirkung: Liefert den ARCUS TANGENS von X im Bogenmass im Bereich -PI/2 bis PI/2. Der Ausdruck X kann von jedem numerischen Typ sein; die Berechnung von ATN erfolgt jedoch auf jeden Fall binaer im Gleitkommaformat.

Beispiel:

```
10 INPUT A
20 PRINT ATN(A)
30 RUN
? 3<RETURN>
 1.24904577
```

3.4.4 CHR$

Format: CHR$(I)

Wirkung: Liefert einen Ein-Byte-String, dessen Element den ASCII-Code I hat. Deshalb muss I im Bereich zwischen 0 und 255 liegen. CHR$ wird gewoehnlich dazu verwendet, spezielle Zeichen oder Steuercodes zu erzeugen.

Beispiele:

PRINT CHR$(147)	Loescht den Bildschirm
PRINT CHR$(66)	Druckt ein B an der Cursorposition

3.4.5 COS

Format: COS(X)

Wirkung: Liefert den COSINUS von X im Bogenmass. Die Berechnung von COS(X) erfolgt binaer im Gleitkommaformat.

Beispiel:

```
10 X=2*COS(.4)
20 PRINT X
RUN
 1.84212199
READY.
```

3.4.6 EXP

Format: EXP(X)

Wirkung: Liefert die X-te Potenz der Zahl e. X muss kleiner oder gleich 88.02969191 sein, sonst wird eine OVERFLOW-Fehlermeldung generiert.

Beispiel:

```
10 X=5
20 PRINT EXP(5-1)
RUN
 54.5981501
READY.
```

3.4.7 FRE

Format: FRE(X)

Wirkung: Liefert die Anzahl der noch nicht benutzten Bytes im BASIC-Programmspeicher. Fuer X kann ein beliebiges Argument angegeben werden, da es keinerlei Wirkung hat. Als Blindargument muss es jedoch vorhanden sein.

Beispiel:

```
PRINT FRE(0)
 1433
READY.
```

3.4.8 INT

Format: INT(X)

Wirkung: Liefert die groesste ganze Zahl, die kleiner oder gleich X ist.

Beispiel:

```
PRINT INT(99.89),INT(-12.11)
 99          -13
READY.
```

3.4.9 LEFT$

Format: LEFT$(X$,I)

Wirkung: Liefert einen String, der aus den I linken Zeichen von X$ besteht. I muss im Bereich zwischen 0 und 255 liegen. Wenn I groesser als die Laenge von X$ ist, wird der gesamte String X$ geliefert. Wenn I Null ist, dann wird ein Leerstring (String der Laenge Null) geliefert.

Beispiel:

```
10 A$="COMMODORE BUEROMASCHINEN"
20 B$=LEFT$(A$,9)
30 PRINT B$
RUN
COMMODORE
READY.
```

3.4.10 LEN

Format: LEN(X$)

Wirkung: Liefert die Anzahl von Zeichen in X$. Es werden alle Zeichen, also auch die nicht abdruckbaren und Leerzeichen gezaehlt.

Beispiel:

```
10 X$="COMMODORE NEU-ISENBURG"
20 PRINT LEN(X$)
RUN
 22
READY.
```

3.4.11 LOG

Format: LOG(X)

Wirkung: Liefert den natuerlichen Logarithmus von X. X muss groesser als Null sein; andernfalls wird eine ILLEGAL QUANTITY-Fehlermeldung ausgegeben.

Beispiel:

```
PRINT LOG(45/7)
 1.86075234
READY.
```

3.4.12 MID$

Format: MID$(X$,I,[,J])

Wirkung: Liefert einen Teilstring von X$ mit J Zeichen, begin- nend beim I-ten Zeichen von X$. I und J muessen im Bereich zwischen 0 und 255 liegen. Wird J weggelassen oder sind rechts vom I-ten Zeichen weniger als J Zeichen in X$ vorhanden, so werden ab dem I-ten Zeichen alle rechten Zeichen von X$ geliefert. Ist I groesser als die Laenge von X$, so wird ein Leerstring (String der Laenge Null) geliefert.

Beispiel:

```
10 A$="GUTEN"
20 B$="MORGEN ABEND MITTAG"
30 PRINT A$;MID$(B$,8,5)
RUN
GUTEN ABEND
READY.
```

3.4.13 PEEK

Format: PEEK(I)

Wirkung: Liefert den Inhalt der Speicherzelle mit der Adresse I als ganzzahligen Wert zwischen 0 und 255. I muss ganzzahlige Werte zwischen 0 und 65535 annehmen. PEEK ist das Gegenstueck zur POKE-Anweisung (s. Abschn. 3.3.23).

Beispiel: A=PEEK(36879)

Der Wert der Variablen A ist nach dieser Anweisung der Code fuer die gegenwaertig eingestellte Hintergrundfarbe des Bildschirms.

3.4.14 POS

Format: POS(X)

Wirkung: Liefert die gegenwaertige Spaltenposition des Cursors auf dem Bildschirm. Die aeusserst linke Spalte ist die Position 0. X ist ein Blindargument, das aus formalen Gruenden jedoch angegeben werden muss.

Beispiel:

```
IF POS(0)20 THEN PRINTCHR$(13)
```

3.4.15 RIGHT$

Format: RIGHT$(X$,I)

Wirkung: Liefert die rechten I Zeichen aus dem String X$. Wenn I gleich oder groesser der Laenge von X$ ist, wird X$ geliefert. Fuer I=0 wird ein Leerstring (String der Laenge Null geliefert). I muss Werte zwischen 0 und 255 einnehmen.

Beispiel:

```
10 A$="COMMODORE BUEROMASCHINEN"
20 PRINT RIGHT$(A$,14)
RUN
BUEROMASCHINEN
READY.
```

3.4.16 RND

Format: RND(X)

Wirkung: Liefert eine Zufallszahl zwischen 0 und 1, die, abhaengig vom Argument X, auf drei verschiedene Weisen generiert werden kann:

X>0: Es wird immer der naechste Wert einer Zufallszahlenreihe, die durch einen numerischen Algorithmus im BASIC-Interpreter berechnet wird, geliefert. Die Reihe ist vom Wert des Argumentes X unabhaengig und wird beim Einschalten des Rechners durch einen zufaelligen Anfangswert initialisiert.

X<0: Jedes Argument X initialisiert eine neue Zufallszahlenreihe. Gleiche Argumente fuehren zu gleichen Zufallszahlenreihen.

X=0: Aus verschiedenen, von einander unabhaengigen Zeitgebern wird durch einen Algorithmus eine Zufallszahl erzeugt.

Beispiel:

```
10 FOR I=1 TO 5
20 PRINT INT(RND(X)*100);
30 NEXT
RUN
 24  30  83  45  1
READY.
```

3.4.17 SGN

Format: SGN(X)

Wirkung: Liefert das Vorzeichen des Argumentes X in folgender codierter Form:

X>0: SGN(X) liefert 1

X=0: SGN(X) liefert 0

X<0: SGN(X) liefert -1

Beispiel:

```
ON SGN(X)+2 GOTO 100,200,300
```

Das Programmm verzweigt nach Zeile 100, wenn der Wert von X negativ ist, nach Zeile 200, wenn er Null ist und nach Zeile 300, wenn er positiv ist.

3.4.18 SIN

Format: SIN(X)

Wirkung: Liefert den SINUS von X im Bogenmass. Die Berechnung von SIN(X) erfolgt binaer in Gleitkommadarstellung. Zwischen SIN(X) und COS(X) besteht der Zusammenhang

COS(X)=SIN(X+3.14159265/2)

Beispiel:

```
PRINT SIN(1.5)
 .997494987
READY.
```

3.4.19 SPC

Format: SPC(I)

Wirkung: Liefert I Leerstellen. SPC kann nur in Verbindung mit der PRINT- oder PRINT#-Anweisung verwendet werden. I muss Werte zwischen 0 und 255 einnehmen.

Beispiel:

```
PRINT"HIER"SPC(15)"DA"
HIER               DA
READY.
```

3.4.20 SQR

Format: SQR(X)

Wirkung: Liefert die Quadratwurzel von X. X muss groesser oder gleich Null sein.

Beispiel:

```
10 FOR X=10 TO 25 STEP 5
20 PRINT X,SQR(X)
30 NEXT
RUN
 10          3.16227766
 15          3.87298335
 20          4.47213595
 25          5
READY.
```

3.4.21 STATUS

Format: STATUS oder ST

Wirkung: Liefert ein Rechnerstatusbyte, dessen Inhalt auf Grund der letzten Ein/Ausgabe-Operation gesetzt wird. Dabei gilt folgende Tabelle:

ST-Bit	ST-Dez.-Aequiv.	Kassette lesen	Ser./IEC-Bus	Kassette VERIFY/LOAD
0	1		Zeitablauf beim Schreiben	
1	2		Zeitablauf beim Lesen	
2	4	kurzer Block		kurzer Block
3	8	langer Block		langer Block
4	16	fataler Lesefehler		fataler Fehler
5	32	Pruefsummenfehler		Pruefsummenfehler
6	64	Fileende	Datenende	
7	-128	Bandende	Geraet nicht angeschl.	Bandende

Beispiel:
```
10 OPEN 6,1,2,"MASTER FILE"
20 GET#6,A$
30 IF ST AND 64 THEN 60
40 ?A$
50 GOTO 20
60 ?A$:CLOSE6
```

Achtung: Das Statuswort fuer die RS-232-Schnittstelle wird im Anhang A.7 detailliert beschrieben.

3.4.22 STR$

Format: STR$(X)

Wirkung: Liefert die Stringdarstellung von X.

Beispiel:
```
10 INPUT"GIB BITTE EINE ZAHL EIN";N
20 PRINT N,LEN(STR$(N))
30 GOTO 10
RUN
GIB BITTE EINE ZAHL EIN? -124<RETURN>
-124     4
GIB BITTE EINE ZAHL EIN? 2<RETURN>
 2       2
GIB BITTE EINE ZAHL EIN? usw.
```

Im zweiten Fall ist die Laenge von STR$(2) deshalb 2, weil in der Stringdarstellung von positiven Zahlen der Zahl immer eine Leerstelle vorangestellt wird.

3.4.23 TAB

Format: TAB(I)

Wirkung: Tabuliert ueber I Spalten in der gegenwaertigen Bildschirmzeile. Steht der Cursor vor der Ausfuehrung von TAB(I) bereits rechts von der I-ten Spalte, so werden I Spalten der Folgezeile uebertabuliert. TAB bezieht sich immer auf den Zeilenanfang in der aeusserst linken Bildschirmspalte (Spalte 0). Die aeusserst rechte Position in einer Zeile ist dann Spalte 21. I muss zwischen Null und 255 liegen. TAB kann nur in Verbindung mit der PRINT- oder PRINT#-Anweisung verwendet werden.

Beispiel:

```
10 PRINT"WARE"TAB(15)"BETRAG":PRINT
20 READ A$,B$
30 PRINT A$TAB(15)B$
40 DATA "BUTTER","DM 2.50"
RUN
WARE            BETRAG

BUTTER          DM 2.50
READY.
```

3.2.24 TAN

Format: TAN(X)

Wirkung: Liefert den TANGENS von X im Bogenmass. Die Berechnung von TAN(X) erfolgt binaer im Gleitkommaformat.

Beispiel:
```
PRINT TAN(5)/2
-1.6902575

READY.
```

3.4.25 TIME

Format: TIME oder TI

Wirkung: Liefert den momentanen Stand der internen Intervall-Uhr, die alle 1/60 Sekunde fortgeschrieben wird. Dies ist keine Echtzeit-Uhr. Die Intervall-Uhr wird beim Einschalten des Rechners initialisiert.

Beispiel:
```
PRINT TI
 154788
READY.
```

3.4.26 TIME$

Format: TIME$ oder TI$

Wirkung: Liefert einen vom Anwender setzbaren und vom System fortgeschriebenen 6-Byte-String in der Anordnung:

HHMMSS

Wird dieser String vom Anwender auf eine bestimmte Uhrzeit gesetzt, so wird diese "Uhr" zeitgerecht vom Rechner fortgeschrieben, bis er abgeschaltet wird.

Beispiel:

```
10 INPUT"BITTE ZEIT (HHMMSS) EINGEBEN";TI$
20 FOR I=1 TO 1000:NEXT
30 PRINT TI$
RUN
BITTE ZEIT (HHMMSS) EINGEBEN? 141223<RETURN>
141234
READY.
```

3.4.27 USR

Format: USR(X)

Wirkung: Verzweigt zu einem Maschinensprache-Unterprogramm, dessen Startadresse vorher in die Zellen mit der Adresse 1 und 2 der zero page (Organisationsspeicher des Interpreters) gespeichert werden muss. Zelle 1 enthaelt den niederwertigen und Zelle 2 den hoeherwertigen Adressteil. Das Argument X wird in einem der Gleitkommaakkumulatoren des Interpreters uebergeben, in den auch das Ergebnis des Unterprogramms abgelegt werden kann, so dass das BASIC-Hauptprogramm dieses Ergebnis unmittelbar einer Variablen zuweisen kann.

Beispiel:
```
10 B=T*SIN(Y)
20 C=USR(B/2)
30 D=USR(B/3)
```

3.4.28 VAL

Format: VAL(X$)

Wirkung: Liefert den numerischen Wert eines Strings, der aus Ziffern besteht. Ausserdem sind die Zeichen . + - und E an den richtigen Stelle erlaubt. Beginnt der String mit einem anderen Zeichen als einer Ziffer, einem Punkt, Plus- oder Minuszeichen, so liefert VAL(X$) Null.

Beispiel:

```
10 X$=".0053"
20 PRINT VAL(X$)
RUN
 5.3E-03

READY.
```

4. Hardware und Betriebssystem des VC20

Im folgenden wird der Aufbau des VC20 anhand von Hardware und Software beschrieben, damit Sie verstehen, wie dieser Rechner organisiert ist und seine und Ihre Programme verarbeitet.

4.1 Aufbau des VC20

Im wesentlichen besteht Ihr VC20 aus dem Mikroprozessor MSC6502, der Betriebssoftware in ROM-Speicherbausteinen (ROM = read only memory = nur lesbarer Speicher), dem Anwender-Programm- und -Datenspeicher-RAM (RAM = random access memory = Speicher fuer wahlfreien Zugriff), den variablen Interface-Bausteinen VIA6522, die den Datenverkehr zu den peripheren Geraeten (Tastatur, Bildschirm, Kassettenstation, Drucker, Floppy Disk usw.) steuern, dem Zeichengenerator-Baustein sowie dem Video-Interface-Chip VIC6561, das die Bild- und Tonsteuerung uebernimmt.
Die grundsaetzliche Arbeitsweise dieser Systemkomponenten werden Sie in den folgenden Abschnitten kennenlernen.

4.1.1 Mikroprozessor

Der Mikroprozessor MCS6502 ist das "Herz" Ihres VC20 und gleichzeitig der komplexeste Baustein auf der Elektronik-Platine. Er steuert saemtliche Operationen des Rechners, indem er die Betriebsprogramme in den ROM-Bausteinen Schritt fuer Schritt abarbeitet. Die Interpretation und Ausfuehrung der einzelnen Befehle erfolgt in den Lese- und Ausfuehrungs-Zyklen des Prozessors. Im Lesezyklus wird ein Programmbefehl in das Befehlsregister (ein 1-Byte-Speicher) des Prozessors geladen. Dann wird der Programmzaehler, der die Adresse dieses Befehls im ROM-Baustein enthaelt, erhoeht, so dass er auf den naechsten auszufuehrenden Befehl zeigt. Im Ausfuehrungszyklus wird der in den Prozessor geladene Befehl interpretiert und die dadurch beschriebene Operation durchgefuehrt. Die Adressen fuer die einzelnen zu uebertragenden Dateneinheiten werden dabei entweder aus dem Befehl selbst hergeleitet oder unter Verwendung von Programmdaten oder Daten aus den internen Registern des Mikroprozessors berechnet.
Diese Steuerungsablaeufe sind als Informationsaustausch ueber den 16-Bit-Adress-Bus, die 8-Bit-Datenrichtungsleitungen sowie die Schreibleitung zu verstehen. Unter einem Bus versteht man mehrere parallele Leitungen, ueber die die Informationen bitparallel gesendet werden. Die Information auf dem Adressbus bestimmt das Ziel, zu den oder von dem Daten uebertragen werden sollen. Die Daten selbst werden auf dem bidirektionalen Datenbus zu und vom Prozessor uebertragen, wobei die Richtung vom Zustand der Schreibleitung bestimmt wird.

4.1.1.1 Ein- und Ausgaenge

Die Ein- und Ausgaenge des Mikroprozessors gliedern sich in 3 Gruppen, von denen jede einen Bus bildet. Ein Bus besteht, wie bereits erwaehnt, aus einem Satz paralleler Leitungen, die die einzelnen Systemkomponenten des Rechners miteinander verbinden.

4.1.1.2 Adressbus

Der Adressbus besteht aus 16 Leitungen und fuehrt die vom Mikroprozessor generierte Adresse zu den Adresseingaengen des Speichers und der Ein/Ausgabe-Einheiten.

4.1.1.3 Datenbus

Der Datenbus besteht aus 8 bidirektionalen Leitungen. Waehrend einer Schreiboperation transportieren diese Leitungen die Daten vom Prozessor zu der durch die Adressleitungen ausgewaehlten Speicherzelle. Bei einer Leseoperation werden die Daten vom Speicher zum Prozessor ueber dieselben Leitungen uebertragen.

4.1.1.4 Steuerbus

Die Funktion der Steuerleitungen, die den Steuerbus bilden, soll an Hand einer einzelnen Leitung beschrieben werden.
Da der Datenbus bidirektionale Datenuebertragung erlaubt, muss der Prozessor eine Moeglichkeit haben, dem Speicher oder den Ein/Ausgabe-Bausteinen mitzuteilen, in welcher Richtung die Daten uebertragen werden, ob die Daten also gelesen oder geschrieben werden sollen. Diese Funktion erfuellt der Schreib/Lese-Ausgang des Prozessors. Liegt an diesem Ausgang eine Spannung, so werden alle Daten vom Speicher oder den Ein/Ausgabe-Bausteinen zum Prozessor uebertragen (Leseoperation). Ist der Schreib/Lese-Ausgang des Prozessors dagegen auf Masse geschaltet, werden Daten vom Prozessor zum Speicher oder den Ein/Ausgabe-Bausteinen uebertragen (Schreiboperation).
Die weiteren Leitungen des Steuerbus' sind die Systemtaktleitung fuer die Steuerung des gesamten Systems in festen Zeitintervallen, die Reset-Leitung, mit deren Hilfe der Prozessor beim Einschalten des Rechners initialisiert wird, sowie die Interruptleitungen, mit deren Hilfe die Abarbeitung eines Programms durch den Prozessor unterbrochen und ein neues Programm bei einer spezifizierten Speicheradresse gestartet werden kann.

4.1.2 Speicher fuer das Betriebssystem (ROM)

Der Betriebssystemspeicher enthaelt alle Programme, die zum Betrieb Ihres VC20 erforderlich sind (Bildschirmeditor, Ein/Ausgabe-Programme, BASIC-Interpreter). Zur Ausfuehrung dieser Programme, die ja aus einer Folge von Befehlen und Daten bestehen, holt sich der Prozessor die einzelnen Befehle, indem er die entsprechende Programmadresse auf den Adressbus legt. Dadurch wird der Befehl aus dem Speicher in Form einer binaeren 8-Bit-Information auf den Datenbus gelegt und zum Prozessor transportiert.

Der Betriebssystemspeicher kann vom Prozessor nur auf die eben beschriebene Weise gelesen werden. Es koennen keine Daten dort hinein geschrieben werden. Andererseits bleibt die Information in diesem Speicher im Gegensatz zum Arbeitsspeicher erhalten, wenn der Rechner ausgeschaltet wird. Es handelt sich hier also um einen nicht-fluechtigen Speicher.

4.1.3 Arbeitsspeicher (RAM)

Der Schreib/Lese-Speicher Ihres VC20 erlaubt die Speicherung von Eingabedaten, arithmetischen Operationen und anderen Datenmanipulationen, kurz, von allen Daten und Programmen, die mit dem VC20 verarbeitet werden sollen. Jede Adresse dieses Speichers bezeichnet eine Gruppe von 8 Speicherzellen (Bits), die zusammen ein Byte bilden. Die Daten in diesen Speicherbytes bleiben jedoch nur solange erhalten, wie der Rechner eingeschaltet ist. Beim Ausschalten wird der gesamte RAM geloescht. Es handelt sich hierbei also um einen fluechtigen Speicher.

4.1.4 Variable Interface-Adapter (VIA)

Die variablen Interface-Adapter stellen die Schnittstellen zwischen Prozessor und Speicher einerseits und der Tastatur, dem Bildschirm sowie dem Userport, Steuerport und dem seriellen Systembus andererseits dar. Der serielle Systembus dient der Kommunikation Ihres VC20 mit peripheren Geraeten wie Drucker oder Floppy Disk. Jedem Anschluss der VIAs ist eine eigene Adresse fuer den Datenaustausch mit dem Mikroprozessor zugeordnet.

4.1.5 Video-Interface-Chip (VIC)

Der Video-Interface-Baustein VIC6561 hat mehrere Funktionen. Er steuert die gesamte Farbvideografik und enthaelt alle Komponenten zur Erzeugung farblich programmierbarer Zeichen mit hoher Bildschirmaufloesung. Ausserdem erlaubt dieser Baustein die Erzeugung von Toenen und Geraeuscheffekten und er enthaelt Analog-Digital-Wandler fuer die Versorgung von Video-Spielen. Das akustische System des VIC besteht aus drei voneinander unabhaengig programmierbaren Tongeneratoren, einem Generator fuer weisses Rauschen sowie einem Amplitudenmodulator. Die Funktion dieses Bausteins wird im Anhang F ausfuehrlich beschrieben.

4.1.6 Zeichengenerator

Der Zeichengenerator erzeugt alle Zeichen, die Ihr VC20 verwendet. Dieser Zeichensatz ist zweimal gespeichert, und zwar fuer die normale und die inverse Zeichendarstellung auf dem Bildschirm. Die Zeichen sind im 6-Bit-ASCII-Code verschluesselt und in Zellen mit 8x8 Bits abgelegt. Der Zugriff auf diese Zeichenmatrizen wird im Abschnitt 4.4 detailliert beschrieben.

4.2 Speicherorganisation

Der Mikroprozessor Ihres VC20 kann bis zu 65536 voneinander unabhaengige Speicherbytes zu je 8 Bit verwalten und adressieren. Sie koennen sich den Speicher als Buch mit 256 Seiten zu je 256 Speicherbytes vorstellen. Diese Seiten werden (in hexadezimaler Schreibweise) von $00 bis $FF gezaehlt. So ist z.B. die Seite $80 der Speicherbereich von 256 Bytes, der von der Adresse $8000 bis zur Adresse $80FF reicht.
Da der Mikroprozessor MCS6502 zur Bildung einer Speicheradresse zwei Bytes verwendet, kann das eine Byte als Seitenzahl und das andere Byte als Platznummer in dieser Seite aufgefasst werden.
Vom Gesamtspeicher kann der aktive Arbeitsspeicher (RAM) beim VC20 3.58 Kilobyte (kByte) (Adresse $1000 (dez. 4096) bis $1DFF (dez. 7679)), 6.65 kByte (Adresse $0400 (dez. 1024) bis $1DFF (dez. 7679)) oder hoechsten 32 kByte durch Hinzufuegen einer 24-kByte-Speichererweiterung einnehmen. Der unterste Speicherbereich von 1 kByte (Adresse $0000 bis $03FF) wird vom Betriebssystem benoetigt und steht dem Anwender daher nicht zur Verfuegung.
Damit verfuegt der VC20 also ueber drei verschiedene Speichertypen:

1. Arbeitsspeicher (RAM)
2. Betriebssystemspeicher (ROM)
3. Ein/Ausgabe-Speicherzellen der VIAs

Diese Speichertypen sowie ihre Organisation sind in der unten stehenden Abbildung noch einmal zusammengestellt. Die einzelnen Speicherbereiche werden in den folgenden Abschnitten beschrieben.

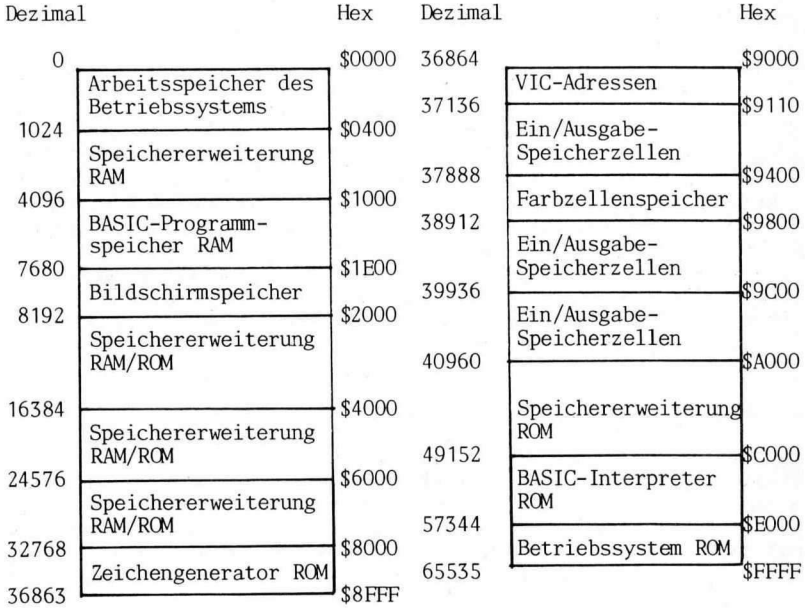


Abb. 4.1: Speicherorganisation des VC20

4.2.1 Arbeitsspeicher und Anwenderprogramme

Die ersten 1024 Bytes des RAM werden vom BASIC-Interpreter und vom Betriebssystem des VC20 als Arbeitsspeicher, Stapelspeicher und Datenpuffer fuer die Kassettenstation benoetigt und stehen Ihnen deshalb nicht als Anwenderspeicher fuer Programme zur Verfuegung. Ab Adresse $1000 (dez. 4096) beginnt der Anwenderspeicher der zusammen mit dem Bildschirmspeicher bis zur Adresse $1FFF (dez. 8191) reicht.

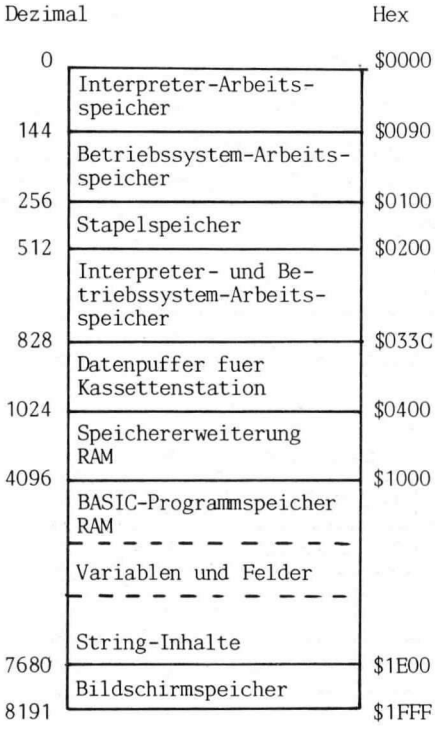


Abb. 4.2: Aufteilung des Arbeitsspeichers

Die Speicherplaetze zwischen $0100 und $01FF (dez. 256 bis 511) werden vom BASIC-Interpreter, vom Betriebssystem und vom Mikroprozessor als Stapelspeicher benoetigt. Dieser Speicher beginnt mit der Adresse $01FF und wird abwaerts dynamisch entsprechend den Anforderungen des Interpreters, des Betriebssystems oder des Mikroprozessors gefuellt oder geloescht. Falls der Stapelzeiger das Ende des Stapelspeichers bei $0100 erreicht, wird eine OUT OF MEMORY-Fehlermeldung ausgegeben.
Der Speicherbereich zwischen $0200 und $033D (dez. 512 bis 827) wird als zusaetzlicher Arbeitsspeicher vom BASIC-Interpreter und vom Betriebssystem verwendet. Die Speicherplaetze $033C bis $03FF (dez. 828 bis 1023) bilden einen Datenpuffer fuer die Kassettenstation.

Der Speicherbereich von $1000 bis $1DFF (dez. 4096 bis 7679) repraesentiert den eigentlichen Anwenderspeicher fuer BASIC-Programme. Dabei werden die Programme ab Adresse $1000 aufwaerts abgelegt. Die waehrend der Programminterpretation erkannten Variablen werden im Anschluss an das Programm abgespeichert. Ggfs. verwendete Felder schliessen sich daran an. Die Inhalte von Stringvariablen, also die Strings selbst, werden bei Adresse $1DFF beginnend abwaerts gespeichert. Wenn die beiden Zeiger fuer Auf- und Abwaertsspeicherung denselben Wert enthalten, wird eine OUT OF MEMORY-Fehlermeldung ausgegeben.

4.2.2 RAM- und ROM-Erweiterungen

Der Adressbereich zwischen $0400 und $0FFF (dez. 1024 bis 4095) ist fuer RAM-Speichererweiterungen vorgesehen. Im Bereich zwischen $2000 und $7FFF (dez. 8192 bis 32767) koennen sowohl RAM- als auch ROM-Erweiterungen eingesetzt werden, waehrend der Bereich zwischen $A000 und $BFFF (dez. 40960 bis 49151) nur fuer ROM-Erweiterungen reserviert ist (s.a. Abb. 4.3).

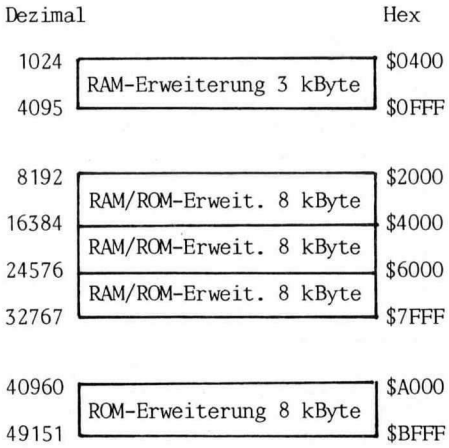


Abb. 4.3: RAM/ROM-Speichererweiterungen

4.2.3 Betriebssystem, Interpreter und Ein/Ausgabe

Die Speicherplaetze zwischen $9110 und $93FF (dez. 37136 bis 37887) sowie zwischen $9800 und $9FFF (dez. 38912 bis 40959) sind die in den Speicherbereich mit einbezogenen Ein/Ausgabe-Speicherplaetze. Im Bereich zwischen $C000 und $DFFF (dez. 49152 bis 57344) ist der BASIC-Interpreter und im Bereich zwischen $E000 und $FFFF (dez. 57345 bis 65535) das VC20-Betriebssystem untergebracht (s.a. Abb. 4.4).

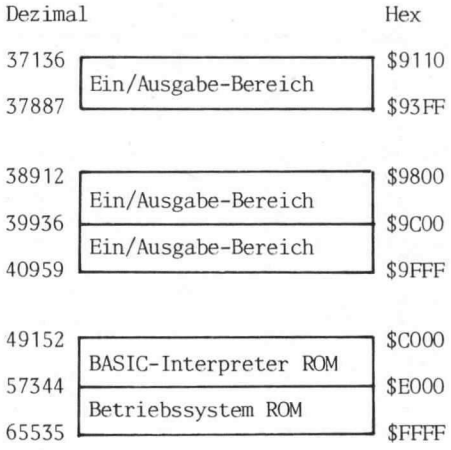


Abb.4.4: Speicherbereiche fuer Ein/Ausgabe, Interpreter und Betriebssystem

4.3 Der BASIC-Interpreter des VC20

Der VC20-BASIC-Interpreter fuehrt ein BASIC-Anwenderprogramm aus, indem er jede in ihrer komprimierten Form gespeicherte Programmzeile interpretiert.
Zunaechst wollen wir kennenlernen, wie ein BASIC-Programm im Speicher abgelegt wird.
Waehrend der Eingabe einer Programmzeile ueber die Tastatur hat der Bildschirmeditor die Kontrolle. Er erlaubt die Editierung der Zeile, bis die RETURN-Taste gedrueckt wird. In diesem Moment geht die Kontrolle an den BASIC-Interpreter ueber. Dieser uebersetzt zunaechst die Zeile in ihre komprimierte Form, indem er alle reservierten Woerter durch 1-Byte-Codes (s. Tabelle C.1 im Anhang C) ersetzt und legt sie dann nach aufsteigender Zeilennummer im Speicher ab. Dazu durchsucht der Interpreter den bisher gefuellten Anwenderspeicher nach dieser Zeilennummer. Findet er eine Zeile mit deselben Nummer, so wird diese Zeile durch die gerade eingegebene ersetzt, andernfalls wird die Zeile hinter der mit der naechsthoeheren bzw. vor der mit der naechstniedrigeren Zeilennummer abgelegt. Die Programmzeilen werden vom Anfang des Anwenderspeichers beginnend (Adresse $1000 oder $0400) gespeichert.

Am Ende eines Programms werden die Variablen und im Anschluss daran die Felder abgelegt. Diese drei Bereiche beginnen bei niedrigeren Adressen und werden zu hoeheren Adressen hin aufgebaut. Die Inhalte von Strings werden dagegen vom Ende des Anwenderspeichers zu niedrigeren Adressen hin abgelegt. Der BASIC-Interpreter baut diese Bereiche im Verlaufe der Programmerstellung und -abarbeitung unter Verwendung von 8 Paaren von Adresszeigern auf (s.a. Abb. 4.5). Jeder Adresszeiger enthaelt eine Adresse in der Anordnung niederwertiges Byte/hoeherwertiges Byte.

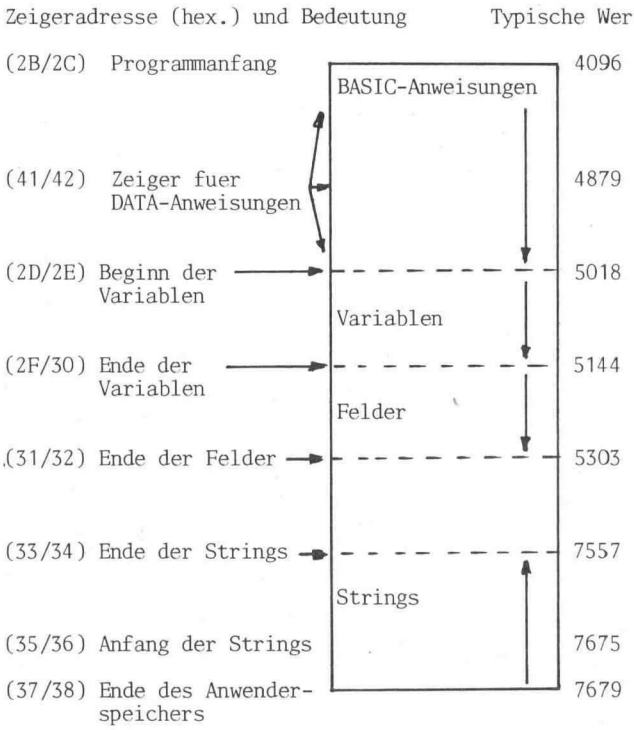


Abb. 4.5: Adresszeiger fuer den Anwenderprogrammbereich

4.3.1 Speicherung von BASIC-Anweisungen

Abb. 4.6 auf der naechsten Seite zeigt das Format, in dem die Programmzeilen vom BASIC-Interpreter im Speicher abgelegt werden. In den Speicherzellen dez. 4096 und 4097 ist die Anfangsadresse der naechsten Programmzeile gespeichert und zwar wie bei allen Adressen in der Weise, dass zuerst das niederwertige Adressbyte (das die Speicherzelle innerhalb einer Speicherseite bezeichnet) und dann das hoeherwertige Adressbyte (das die Speicherseite selbst bezeichnet) abgelegt wird. Die Anfangsadresse der naechsten Zeile enthaelt wieder eine Koppeladresse zur folgenden Zeile usw. Eine Koppeladresse bestehend aus binaeren Nullen ($0000) kennzeichnet das Programmende.

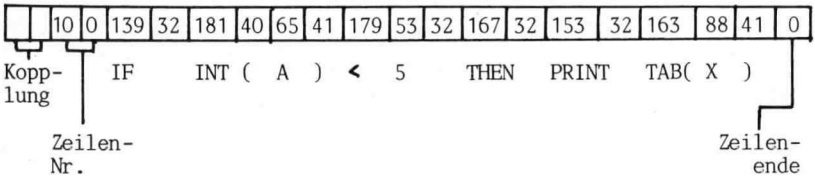
BASIC-Programmzeilen werden nach aufsteigender Zeilennummer abgelegt, obwohl Koppeladressen existieren. Letztere dienen dem schnellen Durchsuchen von Zeilennummern.
An die Koppeladresse schliesst sich die in gleicher Weise gespeicherte Zeilennummer an. Zeilennummern zwischen 0 und 63999 werden in der Form
0,0 bzw. 255,249 dez. gespeichert:

```
                    4096,4097      4098,4099  4100

Programm-:  Koppeladresse  Zeilennr.  komprimierter BASIC-Text  0
anfang      .
            .
            Koppeladresse  Zeilennr.  komprimierter BASIC-Text  0
            .
            .
            .
            .
            Koppeladresse  Zeilennr.  komprimierter BASIC-Text  0

Programm-:  0 0
ende
```

Abb. 4.6: Speicherung von BASIC-Programmzeilen

Nach der Zeilennummer folgt der BASIC-Text der Anweisung(en). Alle reservierten BASIC-Woerter und die mathematischen Operatoren werden in einen 1-Byte-Code komprimiert. Bei diesem Code ist generell das hoechstwertige Bit auf 1 gesetzt, so dass der Code im Bereich zwischen dezimal 128 und 255 liegt. Die anderen Elemente des BASIC-Textes wie Variablennamen, Strings usw. werden im ASCII-Code abgelegt. Tabelle C.1 im Anhang C enthaelt alle Byte-Codes, die in komprimierten BASIC-Zeilen vorkommen koennen. Die Codes werden nach dieser Tabelle interpretiert, es sei denn, sie folgen auf eine ungerade Zahl von Anfuehrungszeichen ("). In diesem Fall werden solche Codes als zum String gehoerige Zeichen aufgefasst. Eine Sonderstellung nehmen die beiden reservierten Woerter TAB und SPC ein. Hier wird die linke Klammer als Bestanddteil des Schluesselwortes aufgefasst, wie in untenstehendem Beispiel dargestellt ist:

```
10 IF INT(A)<5 THEN PRINT TAB(X)
```



4.3.2 Schluesselwort-Codes

Den Operatoren + - * ↑ <=> sowie AND, OR und NOT sind ebenfalls 1-Byte-Codes mit gesetztem hoechsten Bit zugeordnet, da sie fuer den Interpreter die gleiche Bedeutung wie die reservierten Woerter haben. Die entsprechenden ASCII-Codes fuer diese Zeichen werden nur dann abgesetzt, wenn sie als Bestandteil eines Strings auftreten.

4.3.3 Leerstellen in Programmzeilen

Leerstellen in Programmzeilen werden mit Ausnahme der unmittelbar auf die Zeilennummer folgenden grundsaetzlich in ihrem ASCII-Code (dez. 32) mit gepeichert. Beim Listen eines Programmes wird eine Leerstelle zwischen Zeilennummer und erstem Schluesselwort erzeugt. Sie koennen durch Weglassen von Leerstellen in den Programmzeilen erheblich Speicherplatz sparen. Das Programm ist dann jedoch nicht mehr so gut lesbar. Eine weitere Moeglichkeit, Speicherplatz zu sparen, besteht darin, mehrere Anweisungen getrennt durch Doppelpunkt, in eine Programmzeile zu schreiben, da jede Programmzeile 5 Verwaltungsbytes (je zwei fuer die Koppeladresse und Zeilennummer und ein Null-Byte am Zeilenende) benoetigt.

4.3.4 Null-Bytes

Die Laenge einer Programmzeile ist variabel. Deshalb wird zur Kennzeichnung des Zeilenendes jede Programmzeile mit einem Null-Byte (binaere Null) abgeschlossen. Der Wert Null innerhalb einer Programmzeile wird dagegen in seinem ASCII-Code (dez. 48) gespeichert. Diese Nullbytes verwendet der Interpreter bei der Programmausfuehrung als Merkmale, wenn er durch den komprimierten BASIC-Text von links nach rechts geht, die Schluesselwoerter heraussucht und die dadurch bezeichneten Operationen ausfuehrt. Die dem Ende einer Programmzeile folgenden 4 Bytes beinhalten die Koppeladresse und Zeilennummer der Folgezeile. Zum Auffinden dieser Folgezeile wird jedoch nicht das Null-Byte sondern vielmehr die Koppeladresse der gerade interpretierten Zeile verwendet. Drei aufeinanderfolgende Null-Bytes (das Zeilenende der letzten Programmzeile sowie zwei Koppeladressbytes) kennzeichenen das Ende des BASIC-Programmtextes.

4.3.5 Programmformat bei der Kassetten-Speicherung

Programme werden auf Kassette (oder auch Floppy Disk) in demselben Format gespeichert, wie sie im Anwenderspeicher stehen, also in einer kontinuierlichen Folge von Bytes mit Koppeladressen und Null-Bytes.

4.3.6 Programmformat-Kompatibilitaet

Die Verwendung von 1-Byte-Codes anstelle von reservierten Woertern (Schluesselwoerter) ist keine spezielle Eigenart Ihres VC20. Es gibt jedoch keinen standardisierten Code fuer die verschiedenen existierenden BASIC-Interpreter. Aus diesem Grunde ist auch ein auf Kassette gespeichertes Programm in VC20-BASIC nicht kompatibel mit anderen BASIC-Programmen. Andererseits koennen auf anderen Computern erzeugte BASIC-Programme nicht vom VC20-BASIC-Interpreter geladen werden.

4.4 Zugriff auf Farb- und Zeichenmatrizen

Zur programmierten Erzeugung farbiger Zeichen benutzt Ihr VC20 drei verschiedene Speicherbereiche, in denen Zeichen- und Farbzeiger sowie die darzustellenden Zeichen selbst gespeichert sind. Im folgenden werden diese Bereiche sowie der Zugriff auf sie im einzelnen beschrieben.

4.4.1 Zeichenzeiger

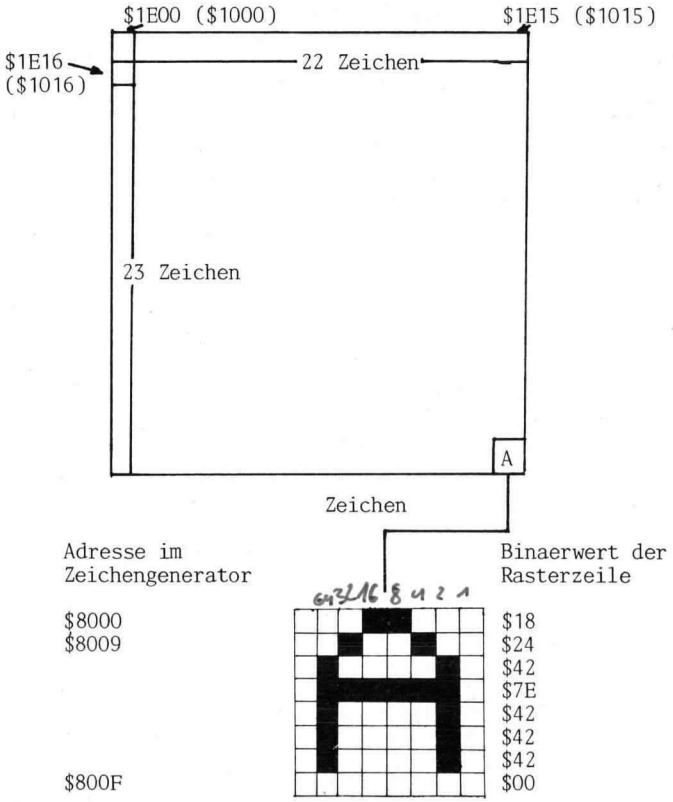
Der Zeichenzeiger-Bereich ist ein Byte-Block im RAM von 506 Bytes, der als Video-Matrix bezeichnet wird. Jedes Byte dieser Matrix zeigt auf ein einzelnes abzubildendes Zeichen.

4.4.2 Anzeige von Zeichen

Der Bereich der darstellbaren Zeichen besteht aus Bloecken von 8 oder 16 Bytes Laenge. Diese Zellen enthalten das abzubildende Punktraster fuer das jeweilige Zeichen und liegen im RAM oder ROM, je nachdem, wie die Zeichen auf den Schirm gebracht werden sollen.

4.4.2.1 Bildschirmspeicher

Dieser RAM-Bereich zwischen $1E00 und $1FFF (dez. 7680 bis 8191) ist fuer die Speicherung der auf dem Bildschirm abzubildenen Zeichen reserviert. Jedes dargestellte Zeichen belegt in diesem Speicher 1 Byte. Das Zeichen selbst ist, wie bereits erwaehnt, in einem 8x8-Bit-Block gespeichert, wie dies in der Abbildung 4.7 auf der naechsten Seite dargestellt ist.



Die in dem oberen Teil der Abbildung in Klammern angegebenen Adressen beziehen sich auf eine vorhandene RAM-Erweiterung von 3 kByte. In diesem Fall legt das Betriebssystem bei der Initialisierung den Bildschirm-Speicher zwischen $1000 und $1F0A an.

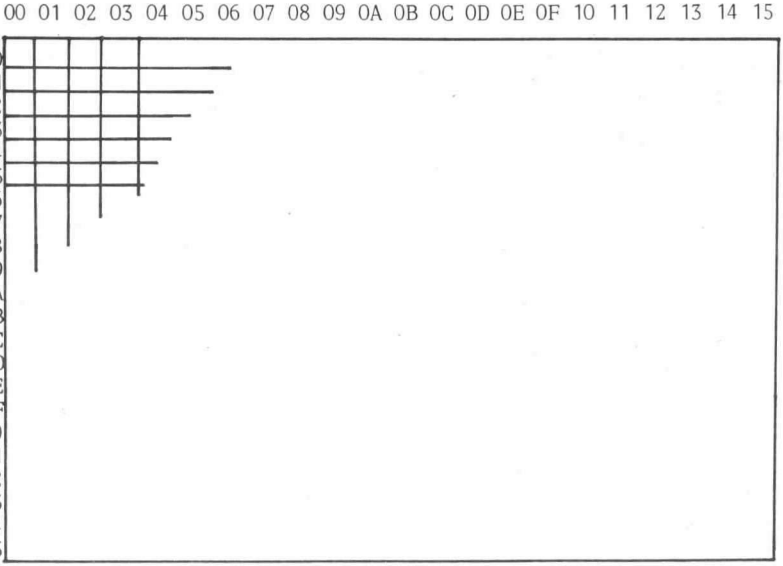
Abb. 4.7: Schema der Bildschirmzellen fuer die Zeichendarstellung

4.4.3 Farbzeiger

Der Speicherbereich fuer die Farbzeiger besteht aus einer Gruppe von Farbzellen im RAM-Bereich $9400 bis $97FF (dez. 37888 bis 38912), der als Farbmatrix bezeichnet wird. Von diesen Farbzellen definieren die jeweils 4 niederwertigen Bits die Farbe, in der die einzelnen Zeichen dargestellt werden sollen und waehlen einen der beiden Farbmodi.

4.4.4 Formatorganisation fuer die Bildschirmanzeige

Der Mikroprozessor organisiert das Format fuer die Video-Matrix, die Farbmatrix und die Zeichenzellen in geeigneter Weise, damit die entsprechenden Daten auf dem Bildschirm dargestellt werden koennen. Abb. 4.8 zeigt das Format der Videomatrix, in der Zeichen in 22 Spalten und 23 Zeilen gespeichert werden koennen. Die Organisation dieser Video-Matrix wird im Anhang F.3 detailliert beschrieben.



Spalten- und Zeilennummern sind hexadezimal angegeben.

Abb. 4.8: Video-Matrix

Es existieren also 506 Speicherzellen fuer die Zeichen, die selbst in einem 8x8-Punktraster abgebildet werden. Damit ergibt sich eine Bildschirmaufloesung von 176 Punkten in der Zeile und 184 Bildpunkten in der Spalte. In jede der 506 Matrixzellen kann z.B. durch Druecken einer Taste ein Zeichenindex fuer das in dieser Position auf dem Bildschirm abzubildende Zeichen gespeichert werden. Aus dem Index wird mit Hilfe des Video-Interface-Chip 6561 eine Adresse errechnet, die auf das in dieser Position abzubildende Zeichen im Zeichengenerator zeigt. Dort wird das Zeichen geholt und in der bezeichneten Position des Bildschirms abgebildet (s.a. Anhang F.3).

Beschreibung der RS-232(V24)-Schnittstelle

A.1 Allgemeines

Die RS-232-Schnittstelle des VC20-Systems setzt sich aus vier Elementen zusammen:

1. Unterstuetzung durch den BASIC-Interpreter.
2. Puffersystem.
3. Byte/Bit-Behandlung.
4. Aeussere Hardware zur Erzeugung der geeigneten Spannungpegel.

Die Programmschnittstelle zum BASIC-Interpreter wird durch die ueblichen BASIC-Anweisungen OPEN, CLOSE, CMD, INPUT#, GET#, PRINT# und STATUS realisiert. Die Verwendung dieser Anweisungen wird an Hand von Beispielen im weiteren Verlauf dieses Anhanges behandelt. Die RS-232-Schnittstelle kann sowohl von BASIC- als auch von Maschinensprache-Programmen (s. Anhang B) bedient werden.
Mit dem Puffersystem sollte sich jeder Programmierer, der die Schnittstelle benutzen moechte, vorher vertraut machen. Das Puffersystem besteht aus zwei 256-Byte-Puffern, die im "first-in-first-out"-Modus betrieben werden, d.h. die Zeichen werden in der Reihenfolge aus dem Puffer ausgegeben, wie sie hineingeschrieben wurde. Das Puffersystem belegt bei Bedarf die obersten beiden Speicher-Seiten des Anwender-Programmspeichers. Durch das Eroeffnen eines RS-232-Kanals mit einer OPEN-Anweisung werden diese 512 Bytes fuer die Puffer reserviert. Ist oberhalb des im Speicher befindlichen BASIC-Programms nicht mehr ausreichend Platz vorhanden, so wird keine Fehlermeldung ausgegeben sondern das Programm wird vielmehr zerstoert. Zeichen werden mit den Anweisungen INPUT# und GET# aus dem Puffer gelesen und mit PRINT# und CMD in den Puffer geschrieben.
Die Byte/Bit-Behandlung fuer die Schnittstelle laeuft unter der Kontrolle von Zeitgebern im VIA6522-Baustein ab und ist interrupt (unterbrechungs-)-gesteuert. Der VIA-Baustein 6522 setzt eine nicht maskierbare Interrupt-Anforderung, die es erlaubt, die RS-232-Schnittstelle im Vordergrund unabhaengig von im Hintergrund ablaufenden BASIC- oder Maschinensprache-Programmen zu betreiben. Die Betriebssystemroutinen zur Bedienung des Kassettengeraetes und der seriellen Systemschnittstelle sind so konzipiert, dass die Datenspeicherung durch die Interruptanforderungen fuer die RS-232-Schnittstelle nicht gestoert werden kann. Deshalb koennen, waehrend das Kassettenlaufwerk oder Peripherie an der seriellen Systemschnittstelle aktiv ist, keine Daten ueber die RS-232-Schnittstelle uebertragen werden.

A.2 Eroeffnen eines RS-232-Datenkanals

Zu jeder Zeit sollte nur ein RS-232-Datenkanal eroeffnet sein. Eine zweite RS-232-OPEN-Anweisung wuerde naemlich die Pufferzeiger zuruecksetzen und die noch nicht ausgegebenen oder empfangenen Daten wuerden dem BASIC-Interpreter verloren gehen. Im Filenamenfeld der OPEN-Anweisung duerfen bis zu vier Zeichen angegeben werden. Die ersten beiden sind die Codes fuer das Kontroll- und Befehlsregister. Die anderen beiden sind fuer zukuenftige Anwendungen (Anwenderspezifizierte Baud-Rate, Satztrennzeichen-Generierung und -Unterdrueckung usw.) vorgesehen. Die Angaben fuer das Kontrollregister werden im Hinblick auf eine nicht implementierte Baud-Rate **nicht** geprueft.

BASIC-Syntax: OPEN#lf,2,0,"<Kontrollregister><Befehlsregister>"

lf: Logische Filenummer (0 < lf < 256). Wird eine Filenummer ueber 127 angegeben, so wird nach jedem logischen Datensatz zusaetzlich zum Wagenruecklaufcode (CHR$(13)) ein Zeilensprung-Code (CHR$(10)) ausgegeben.

<Kontrollregister>: Ein 1-Byte-Zeichen mit einer Bit-Kombination gemaess Abb. A.1 zur Einstellung der Baud-Rate (Bit-Uebertragungsrate).

<Befehlsregister>: Ein 1-Byte-Zeichen mit einer Bit-Kombination gemaess Abb. A.2 zur Einstellung der Paritaetskonventionen sowie der Uebertragungsart.

Achtung: Wenn Sie einen RS-232-Kanal in einem BASIC-Programm verwenden, sollten Sie diesen mit der OPEN-Anweisung eroeffnen, **bevor** Sie irgendeine Variable oder eine DIM-Anweisung verwenden, da durch die OPEN-Anweisung zunaechst die CLR-Anweisung ausgefuehrt wird und dann am oberen Speicherende die beiden Datenpuffer mit insgesamt 512 Byte Laenge angelegt werden. Ist dieser Platz nicht mehr vorhanden, so wird Ihr im Speicher befindliches Programm zerstoert.

In den beiden nachfolgenden Abbildungen A.1 und A.2 wird ein gesetztes Bit mit 1 und ein nicht gesetztes Bit mit 0 bezeichnet:

Bit Nr.:	7	6	5	4	3	2	1	0

Stop-Bits: 7

0: Ein Stop-Bit
1: Zwei Stop-Bits

Datenwort-Laenge (Bit 6, Bit 5):

6	5	
0	0	8 Bits
0	1	7 Bits
1	0	6 Bits
1	1	5 Bits

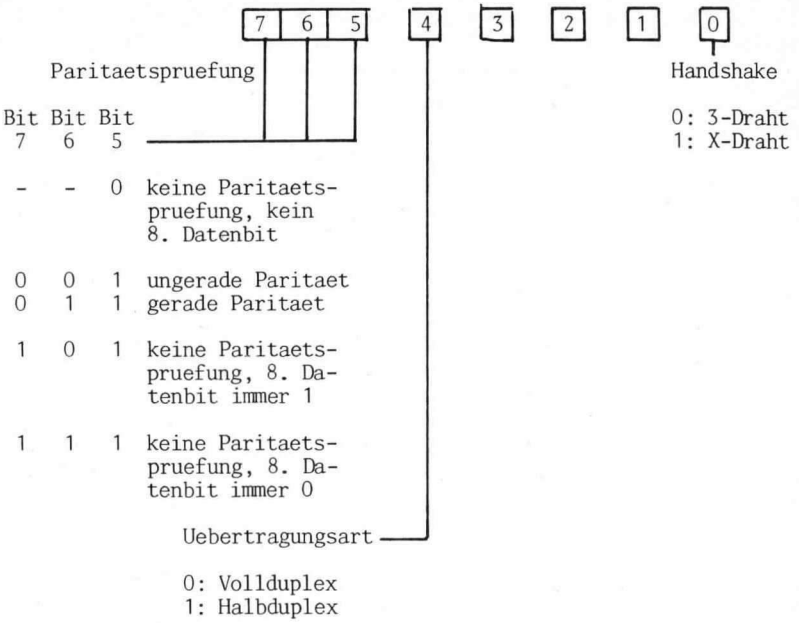
Baud-Rate (Bit 3, Bit 2, Bit 1, Bit 0):

3	2	1	0		
0	0	0	0	Anwender-Rate	(n.i.)
0	0	0	1	50 Baud	
0	0	1	0	75 Baud	
0	0	1	1	110 Baud	
0	1	0	0	134.5 Baud	
0	1	0	1	150 Baud	
0	1	1	0	300 Baud	
0	1	1	1	600 Baud	
1	0	0	0	1200 Baud	
1	0	0	1	1800 Baud	
1	0	1	0	2400 Baud	
1	0	1	1	3600 Baud	(n.i.)
1	1	0	0	4800 Baud	(n.i.)
1	1	0	1	7200 Baud	(n.i.)
1	1	1	0	9600 Baud	(n.i.)
1	1	1	1	19200 Baud	(n.i.)

(n.i.) = noch nicht implementiert

Bit Nr. 4 des Kontrollregisters wird nicht verwendet.

Abb. A.1: Kontrollregister fuer die RS-232-Schnittstelle



Die Bits 3, 2 und 1 des RS-232-Befehlsregisters werden nicht genutzt.

Abb. A.2: Befehlsregister fuer RS-232-Schnittstelle

A.3 Daten aus einem RS-232-Kanal lesen

Beim Einlesen von Daten ueber einen RS-232-Kanal haelt der interne Datenpuffer der Schnittstelle bis zu 255 Zeichen, ehe im Statusbyte, das mit der STATUS-Funktion (s. Abschn. A.7) gelesen werden kann, ein Ueberlauf angezeigt wird. Tritt dies auf, so gehen alle weiteren Daten verloren, solange die Ueberlaufbedingung gesetzt ist. Dehalb ist es angezeigt, bei hoeheren Datenraten Leseroutinen in Maschinensprache zu programmieren.

BASIC-Syntax: GET#lf,<Stringvariable>
INPUT#lf,<Variablenliste>

lf: Logische Filenummer (0<lf<256)

Achtung: Ist die Wortlaenge (Zeichenlaenge) kleiner als 8 Bits, so wird den nicht verwendeten Bits der Wert Null zugewiesen.
Findet GET# keine Daten im Puffer, so liefert diese Anweisung einen Leerstring. INPUT# wartet solange, bis der Puffer ein Nicht-Null-Zeichen enthaelt. Eine Zeichen- kette muss mit einem Wagenruecklauf-Code (CHR$(13)) abgeschlossen sein. Wenn daher waehrend einer INPUT#-Anweisung die CTS- oder DSR-Leitung ("clear to send" und "data set ready") inaktiv werden, so "haengt" sich der Rechner auf und kann nur durch Druecken der RESTORE-Taste neu initialisiert werden. Dann aber sind die Daten verloren. Deshalb sollte die INPUT#-Anweisung in Verbindung mit einem RS-232-Kanal nur verwendet werden, wenn die erforderliche Datenstruktur auch gewaehrleistet ist.

A.4 Daten ueber einen RS-232-Kanal ausgeben

Bei der Datenausgabe ueber einen RS-232-Kanal kann der Datenpuffer bis zu 255 Zeichen halten, ehe im Statusbyte eine Ueberlaufbedingung angezeigt wird. In diesem Fall wartet das System, bis die Uebertragung moeglich ist oder aber die RESTORE-Taste gedrueckt wird.

BASIC-Syntax: CMD lf[,<Liste von Ausdruecken>]
PRINT#lf,<Liste von Ausdruecken>

lf: Logische Filenummer (0<lf<256)

Achtung: Es existiert keine Sendeverzoegerung nach Ausgabe eines Wagenruecklauf-Codes. Es koennen also nur Drucker an der RS-232-Schnittstelle betrieben werden, die ueber einen eigenen Datenpuffer verfuegen. Wurde ein CTS (X-Draht-)-Handshake vereinbart, so wird der Puffer im VC20 gefuellt und die Ausgabe wird dann solange unterbrochen, bis die Uebertragung freigegeben wird.

A.5 Schliessen eines RS-232-Kanals

Beim Schliessen eines RS-232-Kanals werden alle Daten in den Ein- und Ausgabepuffern geloescht, jede Bituebertragung wird angehalten, die RTS- und die Datenausgabeleitung werden inaktiviert und der Speicher, der durch die beiden Puffer belegt wurde, wird wieder freigegeben.

BASIC-Syntax: CLOSE lf

lf: Logische Filenummer (0<lf<256)

Achtung: Ehe die CLOSE-Anweisung ausgefuehrt wird, sollte man sich vergewissern, ob auch wirklich alle Daten uebertragen wurden. Eine Moeglichkeit, dies in BASIC zu realisieren, ist:

```
100 IFST=0 AND (PEEK(37151)AND64)=1 GOTO 100
110 CLOSE lf
```

A.6 Die RS-232-Leitungen

Wie bereits im Abschnitt A.1 dieses Anhanges erwaehnt, wird die RS-232-Schnittstelle durch eine Kombination eines VIA6522-Baussteins mit geeigneter Software in Maschinensprache realisiert. Die einzelnen Leitungen fuer die RS-232-Schnittstelle gemaess des EIA-Standards sowie ihre Pin-Zuordnung am VIA6522 im Adressbereich $9110 bis $911F (dez. 37136 bis 37151) koennen Sie der untenstehenden Tabelle entnehmen:

Tabelle A.1: Die RS-232-Leitungen

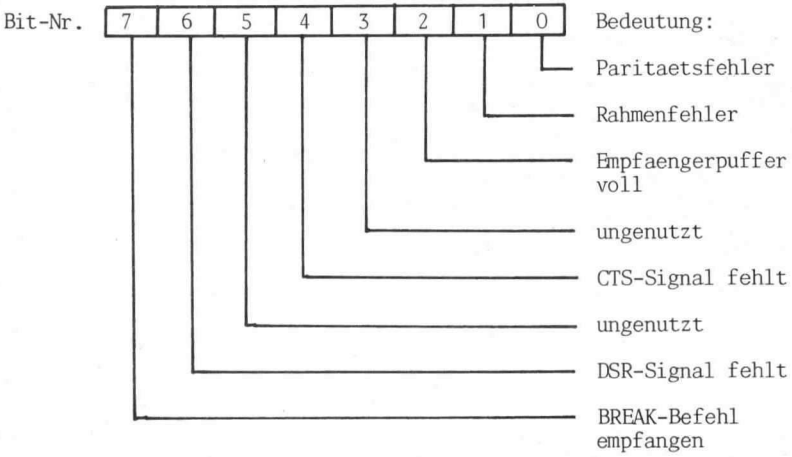
Pin	6522	Beschreibung	EIA	ABV	IN/OUT	Modi
C	PB0	Received data	(BB)	Sin	IN	1 2
D	PB1	Request to send	(CA)	RTS	OUT	1*2
E	PB2	Data terminal ready	(CD)	DTR	OUT	1*2
F	PB3	Ring indicator	(CE)	RI	IN	3
H	PB4	Received line signal	(CF)	DCD	IN	2
J	PB5	Nicht belegt			IN	3
K	PB6	Clear to send	(CB)	CTS	IN	2
L	PB7	Data set ready	(CC)	DSR	IN	2
B	CB1	Received data	(BB)	Sin	IN	1 2
M	CB2	Transmitted data	(BA)	Sout	OUT	1 2
A	GND	Protective ground	(AA)	GND		1 2
N	GND	Signal ground	(AB)	GND		2 3

Modi: 1: 3-Draht-Interface. 2: X-Draht-Interface.
3: Anwender

*: Diese Leitungen liegen beim 3-Draht-Interface hoch.

A.7 Das RS-232-Statusregister

Jeder Ein/Ausgabeverkehr Ihres VC20 wird vom Betriebssystem ueberwacht und eventuelle Fehler oder Besonderheiten werden im Statusregister ST (s.a. Abschn. 3.4.20) vermerkt. Das gilt auch fuer den Datenverkehr ueber die RS-232-Schnittstelle. Deren Statusregister kann ebenfalls durch die STATUS-Funktion der Systemvariablen ST zugeordnet werden. Es ist jedoch hierbei zu beachten, dass durch jedes Lesen des Statusregisters dieses geloescht wird. Bei verschiedenen aufeinanderfolgenden Abfragen muss deshalb die Systemvariable ST zunaechst einer anderen freien Variablen zugeordnet werden (z.B. SR=ST). Nur wenn die letzte Ein/Ausgabeoperation ueber einen RS-232-Kanal lief, wird das RS-232-Statusregister durch diese Operation geloescht. Die Zuordnung der einzelnen Bits des RS-232-Statusregister koennen Sie der untenstehenden Abbildung entnehmen.



Ein Bit-Wert von 0 bedeutet "kein Fehler"

Abb. A.3: RS-232-Statusregister

A.8 BASIC-Beispielprogramm fuer RS-232-Schnittstelle

Das auf der naechsten Seite aufgelistete BASIC-Programm steuert den bidirektionalen Datenverkehr zwischen einem Sichtgeraet "SILENT 700" und einem VC20-Rechner. Dabei wurde folgende Betriebsart gewaehlt:

* Uebertragungsrate 300 Baud (Bit/s)
* 7-Bit-ASCII
* 1 Stop-Bit
* keine Paritaetspruefung
* 8. Datenbit immer 1
* Voll-Duplex
* 3-Draht-Handshake

```
100 OPEN 2,2,0,CHR$(6+32)+CHR$(32+128): REM KANAL WIRD EROEFFNET
110 GET#2,A$: REM EMPFAENGERKANAL INITIALISIEREN
120 REM HAUPTSCHLEIFE
130 GET B$:IF B$="" THEN 130: REM ZEICHEN VON VC20-TASTATUR HOLEN
140 PRINT#2,B$;: REM ZEICHEN AN TERMINAL SENDEN
150 GET#2,C$: REM ZEICHEN VOM TERMINAL HOLEN
160 PRINT B$;C$;: REM ALLE EINGABEN AUF VC20-BILDSCHIRM DRUCKEN
170 SR=ST:IF SR=0 THEN 120: REM FALLS KEINE FEHLER, WEITERMACHEN
180 REM FEHLERAUSGABE
190 PRINT "FEHLER: ";
200 IF SR AND 1 THEN PRINT "PARITAETS-FEHLER"
210 IF SR AND 2 THEN PRINT "RAHMEN-FEHLER"
220 IF SR AND 4 THEN PRINT "EMPFAENGERPUFFER VOLL"
230 IF SR AND 128 THEN PRINT "BREAK-SIGNAL EMPFANGEN"
240 IF (PEEK(37151)AND64)=1 THEN 240: REM UEBERTRAGUNGSENDE ABWART.
250 CLOSE 2:END
```

A.9 Empfaenger/Sender-Pufferzeiger

Das Betriebssystem fuehrt fuer den Datenaustausch ueber die RS-232-Schnittstelle zwei Pufferzeiger von je zwei Bytes, in die die jeweiligen Anfangsadressen von Ein- und Ausgabe-Puffer eingetragen werden, sobald die OPEN-Anweisung ausgefuehrt wird. Bei der CLOSE-Anweisung wird in das jeweils hoeherwertige Byte Null geschrieben, um die Freigabe des Speicherbereiches zu kennzeichnen:

Adresse	symbol. Name	Bedeutung
$00F7/$00F8	RIBUF	Zeiger auf Anfangsadresse des Eingabe-puffers
$00F9/$00FA	ROBUF	Zeiger auf Anfangsadresse des Ausgabe-puffers

Diese Zeiger koennen auch von Maschinenspracheprogrammen oder mittels der POKE-Anweisung von BASIC-Programmen gesetzt werden, um Datenpuffer fuer andere Zwecke zu deklarieren.

A.10 Zero-Page-Speicherplaetze fuer die RS-232-Software

Das Betriebssystem des VC20 benutzt fuer die Steuerung der RS-232-Schnittstelle einige Speicherplaetze der "zero page". Die zero page ist die Seite Nr. 0 des Speichers im Adress-Bereich $0000 bis $00FF (dez. 0 bis 255). Diese Plaetze werden jedoch nur lokal genutzt, da die RS-232-Software interrupt-gesteuert ist. Sie duerfen nicht fuer andere Zwecke vom Anwender benutzt werden. Dennoch sind sie der Vollstaendigkeit halber auf der folgenden Seite zusammengestellt.

Adresse	symbol. Name	Bedeutung
$00A7	INBIT	Bit-Zwischenspeicher bei der Eingabe
$00A8	BITCI	Bit-Zaehler bei der Eingabe
$00A9	RINONE	Flag fuer Startbit-Check bei der Eingabe
$00AA	RIDATA	Zwischenpuffer fuer den Byteaufbau (Eingabe)
$00AB	RIPRTY	Paritaetsbit-Speicher bei der Eingabe
$00B4	BITTS	Bit-Zaehler bei der Ausgabe
$00B5	NXTBIT	naechstes zu uebertragendes Bit (Ausgabe)
$00B6	RODATA	Zwischenpuffer fuer Bytezerlegung (Ausgabe)

Routinen des Betriebssystems

B.1 Uebersicht

Dieser Anhang enthaelt die Beschreibung der Routinen oder Unterprogramme, die das Betriebssystem Ihres VC20 zur Kontrolle aller Rechnerfunktionen verwendet. Alle Routinen, die in der untenstehenden Tabelle B.1 zusammengestellt sind, koennen von Maschinenspracheprogrammen, die Sie selbst erstellt haben, wie Unterprogramme aufgerufen werden. Die Seitenangabe in der Tabelle bezieht sich auf die detaillierte Beschreibung jeder einzelnen Routine im Abschnitt B.3:

Tabelle B.1: Aufrufbare Betriebssystem-Unterprogramme

Symbol. Name	Einsprung-Adresse	Funktion	Seite
	64309		
ACPTR	$FFA5	Byteeingabe ueber IEEE-Bus	140
CHKIN	$FFC6	Eingabekanal eroeffnen	141
CHKOUT	$FFC9	Ausgabekanal eroeffnen	142
CHRIN	$FFCF	Byteeingabe ueber Kanal	143
CHROUT	$FFD2	Byteausgabe ueber Kanal	144
CIOUT	$FFA8	Byteausgabe ueber IEEE-Bus	145
CLALL	$FFE7	Alle Files schliessen	146
CLOSE	$FFC3	logischen File schliessen	147
CLRCHN	$FFCC	Ein- und Ausgabekanaele schliessen	148
GETIN	$FFE4	Zeichen von Eingabegeraet holen	149
IOBASE	$FFF3	Basis-Ein/Ausgabeadresse liefern	150
LISTEN	$FFB1	LISTEN-Befehl an IEEE-Geraet	151
LOAD	$FFD5	RAM von peripherem Geraet laden	152
MEMBOT	$FF9C	Lesen/Setzen Speicheranfangsadresse	153
MEMTOP	$FF99	Lesen/Setzen Speicherendadresse	154
OPEN	$FFC0	logischen File eroeffnen	155
PLOT	$FFF0	Lesen/Setzen Cursor-X/Y-Position	156
RDTIM	$FFDE	Systemuhr lesen	157
READST	$FFB7	Ein/Ausgabe-Statusbyte lesen	158
RESTOR	$FF8A	Ein/Ausgabe-Zeiger voreinstellen	159
SAVE	$FFD8	RAM auf peripheres Geraet schreiben	160
SCNKEY	$FF9F	Tastatur abfragen	161
SCREEN	$FFED	X/Y-Bildschirmorganisation liefern	162
SECOND	$FF93	Sekundaeradresse ausgeben	163
SETLFS	$FFBA	log., Prim.-, Sekundaeradr. setzen	164
SETMSG	$FF90	Meldungen d. Betr.-Systems absetzen	165
SETNAM	$FFBD	Filenamen-Information absetzen	166
SETTIM	$FFDB	Systemuhr setzen	167
SETTMO	$FFA2	Zeitablauf bei IEEE-Verkehr setzen	168
STOP	$FFE1	STOP-Taste abfragen	169
TALK	$FFB4	TALK-Befehl an IEEE-Geraet	170
TKSA	$FF96	Sek.-Adr. nach TALK-Befehl senden	171
UDTIM	$FFEA	Systemuhr fortschreiben	172
UNLSN	$FFAE	UNLISTEN-Befehl an IEEE-Geraet	173
UNTLK	$FFAB	UNTALK-Befehl an IEEE-Geraet	174
VECTOR	$FF8D	Lesen/Setzen gericht. Ein/Ausgabe	175

B.3 Beschreibung der Betriebssystem-Unterprogramme

Die folgenden Begriffe werden bei der Beschreibung der einzelnen Unterprogramme verwendet:

Funktionsname: Dies ist ein symbolischer Name, der der Einsprungadresse zugeordnet ist und nur mnemonische Bedeutung hat. Der Anwender kann in seinen Assembler-Programmen auch andere Namen verwenden.

Einsprungadresse: Dies ist die Adresse in hexadezimaler Schreibweise, mit der das Unterprogramm aufgerufen wird.

Uebergaberegister: Die hier angegebenen Register des Mikroprozessors dienen der Parameteruebergabe zwischen Anwender- und Unterprogramm.

Vorbereitungsroutinen: Gelegentlich muessen Daten fuer Unterprogramme aufbereitet werden. Dies geschieht mit den unter dieser Rubrik angegebenen Routinen.

Fehleranzeigen: In einigen Faellen bedeutet ein Aussprung aus einem Unterprogramm mit gesetzter Carry-Flag, dass der Akkumulator (Register A) die Nummer eines Fehlers enthaelt, der waehrend der Verarbeitung aufgetreten ist. Die jeweils moeglichen Nummern werden unter dieser Rubrik angegeben.

Stapelbedarf: Hier wird die Anzahl der vom Unterprogramm benoetigten Bytes des Stapelspeichers angegeben.

Beschreibung: Eine kurze Beschreibung soll die Funktion des jeweiligen Unterprogramms fuer Sie verstaendlich machen.

Beispiel: Die Angabe eines kleinen Beispiels in der mnemonischen Schreibweise der Assembler-Sprache fuer der Mikroprozessor MCS6502 soll Ihnen die Anwendung der Unterprogramme erleichtern.

Funktionsname: ACPTR

Einsprungadresse: $FFA5

Uebergaberegister: A

Vorbereitungsroutinen: TALK, TKSA

Fehleranzeigen: Siehe READST

Stapelbedarf: 13

Beschreibung: Diese Routine uebernimmt ein Datenbyte mit Handshake vom IEEE-Bus und uebergibt es im Akkumulator. Vorher muss das Geraet mit der Routine TALK als talker adressiert und ggfs. mit TKSA mit einer Sekundaeradresse versorgt worden sein.

Beispiel:

```
JSR     ACPTR
STA     DATA
```

Funktionsname: CHKIN

Einsprungadresse: $FFC6

Uebergaberegister: X

Vorbereitungsroutinen: OPEN

Fehleranzeigen: 3, 5, 6

Stapelbedarf: 11

Beschreibung: Ein Eingabekanal wird eroeffnet. Wenn ein logischer File mit OPEN eroeffnet wurde, so kann ihm abhaengig von den Charakteristika des Eingabegeraetes ein Eingabekanal zugeordnet werden. Dieses Unterprogramm muss aufgerufen werden, ehe die Unterpro-gramme CHRIN oder GETIN fuer andere Geraete als die Tastatur aufgerufen werden. Bei Eingabe von der Tastatur ohne Zusammenhang mit einem eroeffneten logischen File braucht CHKIN nicht aufgerufen zu werden. Bei IEEE-Geraeten sendet diese Routine einen TALK-Befehl zusammen mit einer ggfs. in der OPEN-Routine angegebenen Sekundaeradresse.

Beispiel:

```
;EROEFFNE KANAL 2 FUER EINGABE
        LDX  #2
        JSR  CHKIN
```

Funktionsname: CHKOUT

Einsprungadresse: $FFC9

Uebergaberegister: X

Vorbereitungsroutinen: OPEN

Fehleranzeigen: 3, 5, 7

Stapelbedarf: 11

Beschreibung: Ein Ausgabekanal wird eroeffnet. Wenn ein logischer File mit OPEN eroeffnet wurde, so kann ihm abhaengig von den Charakteristika des Ausgabegeraetes ein Ausgabekanal zugeordnet werden. Dieses Unterprogramm muss aufgerufen werden, ehe das Unterprogramm CHROUT fuer andere Ausgabegeraete als den Bildschirm aufgerufen wird. Bei Ausgabe auf den Bildschirm ohne Zusammenhang mit einem eroeffneten logischen File braucht CHROUT nicht aufgerufen zu werden. Bei IEEE-Geraeten sendet diese Routine einen LISTEN-Befehl zusammen mit einer ggfs. in der OPEN-Routine angegebenen Sekundaeradresse.

Beispiel:

```
;EROEFFNE KANAL 3 FUER AUSGABE
      LDX  #3
      JSR  CHKOUT
```

Funktionsname: CHRIN

Einsprungsadresse: $FFCF

Uebergaberegister: A

Vorbereitunsroutinen: Keine

Fehleranzeigen: Siehe READST

Stapelbedarf: 3

Beschreibung: Einlesen eines Bytes ueber einen Eingabekanal, der entweder der vom System vorgewaehlte (Tastatur) ist, oder durch CHKIN eroeffnet wurde. Das Datum wird im Akkumulator uebergeben. Der Kanal bleibt nach Rueckkehr aus dem Unterprogramm offen. Ist die Tastatur das Eingabegeraet, so blinkt der Cursor nach dem Druecken einer Taste in der naechsten Schreibposition, bis die RETURN-Taste gedrueckt wurde. Jeder Aufruf dieser Routine uebergibt dann ein Zeichen der eingegebenen Zeichenkette einschliesslich des Wagenruecklauf-Codes.

Beispiel:

```
JSR CHRIN
STA DATA
```

Funktionsname: CHROUT

Einsprungadresse: $FFD2

Uebergaberegister: A

Vorbereitungsroutinen: Keine

Fehleranzeigen: Siehe READST

Stapelbedarf: 3

Beschreibung: Ausgeben eines Bytes ueber einen Ausgabekanal, der entweder der vom System vorgewaehlte (Bildschirm) ist, oder durch CHKOUT eroeffnet wurde. Das Byte kann an verschiedene Geraete am IEEE-Bus ausgegeben werden, wenn nach der zugehoerigen Kanaleroeffnung kein CLRCHN aufgerufen wurde.

Beispiel:

```
;DARSTELLUNG DER BASIC-ANWEISUNG CMD4,"A";
      LDX  #4     ;LOGISCHER FILE 4
      JSR CHKOUT ;AUSGABEKANAL OEFFNEN
      LDA #'A
      JSR CHROUT ;ZEICHEN AUSGEBEN
```

Funktionsname: CIOUT

Einsprungadresse: $FFA8

Uebergaberegister: A

Vorbereitungsroutinen: LISTEN, äSECONDü

Fehleranzeigen: Siehe READST

Stapelbedarf: 1

Beschreibung: Byteausgabe ueber IEEE-Bus. Falls das Ausgabegeraet nicht vorher durch LISTEN adressiert wurde, wird eine Zeitablauf-Fehlerbedingung gesetzt. Diese Routine puffert immer ein Byte zwischen, das durch die UNLSN-Routine zusammen mit dem EOI-Signal vor dem eigentlichen UNLISTEN-Befehl an das Ausgabegeraet gesendet wird.

Beispiel:

```
LDA  #'A
JSR  CIOUT
JSR  UNLSN
```

Funktionsname:	CLALL
Einsprungadresse:	$FFE7
Uebergaberegister:	Keine
Vorbereitungsroutinen:	Keine
Fehleranzeigen:	Keine
Stapelbedarf:	11
Beschreibung:	Alle Files werden geschlossen, indem die Zeiger der Tabelle fuer eroeffnete Files sowie die Ein/Ausgabekanaele durch Aufruf der Routine CLRCHN rueckgesetzt werden.

Beispiel:

```
;DEFINIERTEN AUSGANGSZUSTAND BEI
;PROGRAMMBEGINN HERSTELLEN
      JSR  CLALL  ;ALLE FILES SCHLIESSEN
      JMP  RUN    ;PROGRAMM BEGINNEN
```

Funktionsname: CLOSE

Einsprungadresse: $FFC3

Uebergaberegister: A

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 11

Beschreibung: Ein logischer File wird geschlossen, wenn dieses Unterprogramm nach Abschluss aller Ein- oder Ausgaben mit der logischen Filenummer im Akkumulator aufgerufen wird, die beim OPEN verwandt wurde.

Beispiel:

```
;ERSATZ DER BASIC-ANWEISUNG CLOSE 15
      LDA #15
      JSR CLOSE
```

Funktionsname:	CLRCHN
Einsprungadresse:	$FFCC
Uebergaberegister:	Keine
Vorbereitungsroutinen:	Keine
Fehleranzeigen:	Keine
Stapelbedarf:	9
Beschreibung:	Dieses Unterprogramm schliesst nach abgeschlossenen Ein/Ausgabe-Operationen alle eroeffneten Kanaele und setzt Tastatur und Bildschirm wieder als vorgewaehlte Kanaele 0 und 3. Falls das Ein- oder Ausgabegeraet ein IEEE-Geraet ist, wird durch diese Routine ein UNTALK- oder UNLISTEN-Befehl gesendet, wodurch der Kanal wieder freigegeben wird.
Beispiel:	JSR CLRCHN

Funktionsname: GETIN

Einsprungadresse: $FFE4

Uebergaberegister: A

Vorbereitungsroutinen: äOPEN, CHKINü

Fehleranzeigen: Siehe READST

Stapelbedarf: 11

Beschreibung: Ein Zeichen wird aus dem Eingabepuffer eines Eingabegeraetes gelesen und im Akkumulator uebergeben. Falls das Eingabegeraet ein anderes als die Tastatur ist, muss vorher ein Eingabekanal eroeffnet werden. Im Falle der Tastatur wird ein Zeichen aus dem Tastaturpuffer gelesen. Dieser Puffer erhaelt die Zeichen durch eine Interrupt-Tastatur-Abfrage (s. SCNKEY).

Beispiel:

```
;AUF GEDRUECKTE TASTE WARTEN
WAIT  JSR GETIN
      CMP #0
      BEQ  WAIT
```

Funktionsname: IOBASE

Einsprungadresse: $FFF3

Uebergaberegister: X,Y

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Dieses Unterprogramm liefert die Adresse der Speicherseite, in der Ein/Ausgabe-Speicherplaetze enthalten sind, in den beiden Registern X und Y. Damit kann in Verbindung mit einem Adress-Offset auf speicherbezogene Ein/Ausgabeeinheiten innerhalb des VC20 zugegriffen werden.

Beispiel:

```
JSR  IOBASE
STX  POINT
STY  POINT+1
LDA  #0
LDY  #2          ;ADRESS-OFFSET
STA  (POINT),Y
```

Funktionsname: LISTEN

Einsprungadresse: $FFB1

Uebergaberegister: A

Vorbereitungsroutinen: Keine

Fehleranzeigen: Siehe READST

Stapelbedarf: 8

Beschreibung: Ein LISTEN-Befehl wird zusammen mit dem ATN-Signal an ein IEEE-Geraet gesendet, wobei vorher der Akkumulator mit dessen Adresse zwischen 0 und 30 geladen werden muss.

Beispiel:

```
;LISTEN-BEFEHL AN IEEE-GERAET NR. 8
      LDA  #8
      JSR  LISTEN
```

Funktionsname: LOAD

Einsprungadresse: $FFD5

Uebergaberegister: A, X, Y

Vorbereitungsroutinen: SETLFS, SETNAM

Fehleranzeigen: 0, 4, 5, 8, 9

Stapelbedarf: 11

Beschreibung

Von einem peripheren Eingabegeraet wird eine kontinuierliche Folge von Bytes in den RAM geladen. Aufruf mit A=0 bedeutet "Laden", mit A=1 "Verifizieren". X und Y beinhalten die Adresse, ab der in den RAM geladen werden soll, wenn bei SETLFS als Sekundaeradresse (SA) 3 gegeben wurde. Wurde SA in SETLFS mit 0, 1 oder 2 gesetzt, so wird der RAM ab der Adresse geladen, die im Kopf des zu ladenden Files steht. Bei Rueckkehr enthalten X und Y die Adresse des letzten durch LOAD beschriebenen RAM-Bytes.

Beispiel:

```
        LDA  DEVICE
        LDX  FILNO
        LDY  SA
        JSR  SETLFS
        LDA  #NAME1-NAME
        LDX  #<NAME
        LDY  #>NAME
        JSR  SETNAM
        LDA  #0     ;LOAD-FLAG
        LDX  #$FF   ;VOREINGEST. RAM-BEGINN
        LDY  #$FF
        JSR  LOAD
        STX  VARTAB
        STY  VARTAB+1
        JMP  START
NAME    .BYT 'FILE NAME'
NAME1
```

Funktionsname: MEMBOT

Einsprungadresse: $FF9C

Uebergaberegister X, Y

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Wird dieses Unterprogramm mit gesetztem Carry-Bit aufgerufen, so wird der Zeiger auf die niedrigste RAM-Adresse gelesen und beim Ruecksprung im X- und Y-Register uebergeben. Der voreingestellte Wert ist $0400 oder $1000, falls RAM-Erweiterung eingebaut ist. Wird diese Routine mit geloeschtem Carry-Bit aufgerufen, so wird der Inhalt von X und Y in den genannten Adresszeiger uebertragen.

Beispiel:

```
;SPEICHERANFANG UM 1 SEITE ERHOEHEN
        SEC
        JSR     MEMBOT
        INY
        CLC
        JSR     MEMBOT
```

Funktionsname: MEMTOP

Einsprungadresse: $FF99

Uebergaberegister: X, Y

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Wird dieses Unterprogramm mit gesetztem Carry-Bit aufgerufen, so wird der Zeiger auf die hoechste RAM-Adresse gelesen und beim Ruecksprung im X- und Y-Register uebergeben. Wird das Unterprogramm mit geloeschtem Carry-Bit aufgerufen, so wird der Inhalt des X- und Y-Registers in den genannten Adresszeiger uebertragen.

Beispiel:

```
;FREIGABE DES KASSETTENPUFFERS
        SEC
        JSR     MEMTOP
        TXA
        CLC
        ADC     #192
        BCC     NOINC
        INY
NOINC   TAX
        CLC
        JSR     MEMTOP
```

Funktionsname: OPEN

Einsprungadresse: $FFC0

Uebergaberegister: Keine

Vorbereitungsroutinen: SETLFS, SETNAM

Fehleranzeigen: 1, 2, 4, 5, 6

Stapelbedarf: 2

Beschreibung: Ein logischer File wird eroeffnet. Es werden keine Parameter an die Routine uebergeben. Die beiden Routinen SETLFS und SETNAM muessen jedoch vorher aufgerufen werden.

Beispiel:

```
;ERSATZ DER BASIC-ANWEISUNG
;OPEN15,8,15,"IO"
        LDA     #NAME2-NAME   ;NAMEN-LAENGE
        LDX     #NAME
        LDY     #NAME
        JSR     SETNAM
        LDA     #15
        LDX     #8
        LDY     #15
        JSR     SETLFS
        JSR     OPEN
NAME    .BYTE   "IO"
NAME2
```

Funktionsname: PLOT

Einsprungadresse: $FFF0

Uebergaberegister: X, Y

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Wird dieses Unterprogramm mit gesetztem Carry-Bit aufgerufen, so wird die gegenwaertige Spalten/Zeilenposition des Cursors auf dem Bildschirm gelesen und im X- und Y-Register uebergeben. Ein Aufruf mit geloeschtem Carry-Bit setzt den Cursor auf eine dem Inhalt von X und Y entsprechende Spalten/Zeilenposition auf dem Bildschirm.

Beispiel:

```
;CURSOR IN SPALTE 9, ZEILE 5 SETZEN
        LDX     #9
        LDY     #5
        CLC
        JSR     PLOT
```

Funktionsname: RDTIM

Einsprungadresse: $FFDE

Uebergaberegister: A, X, Y

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Die Systemuhr wird ausgelesen. Dies kann zu jeder Zeit geschehen. Es werden 3 Bytes als 24-Bit-Binaerzahl uebergeben. Der Wert dieser Zahl ist die Zeit in 1/60 Sekunden, die seit dem Einschalten des Rechners oder dem letzten Setzen der Uhr verstrichen ist. Der Akkumulator enthaelt die hochwertigen, das X-Register die mittelwertigen und das Y-Register die niederwertigen 8 Bits dieser Binaerzahl.

Beispiel:

```
        JSR     RDTIM
        STY     TIME
        STX     TIME+1
        STA     TIME+2
        .
        .
        .
TIME    *=*+3
```

Funktionsname: READST

Einsprungadresse: $FFB7

Uebergaberegister: A

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Der gegenwaertige Ein/Ausgabestatus wird uebergeben. Dieses Unterprogramm wird gewoehnlich nach jeder Ein/Ausgabe-Operation aufgerufen, um eventuelle Fehler zu diagnostizieren. Die Bedeutung der einzelnen Bits in dem im Akkumulator uebergebenen Byte ist in Abschnitt 3.4.20 fuer den Kassetten-, IEEE-Bus- und seriellen Bus-Betrieb sowie im Anhang A.7 fuer die RS-232-Schnittstelle detailliert beschrieben. Enthaelt der Akkumulator bei Rueckkehr aus dieser Routine 0, so ist kein Fehler aufgetreten.

Beispiel:

```
;AUF FILEENDE PRUEFEN
        JSR     READST
        AND     #64     ;EOF-BIT PRUEFEN
        BNE     EOF     ;FALLS GESETZT
```

Funktionsname:	RESTOR
Einsprungadresse:	$FF8A
Uebergaberegister:	Keine
Vorbereitungsroutinen:	Keine
Fehleranzeigen:	Keine
Stapelbedarf:	2
Beschreibung:	Alle Adresszeiger des Betriebssystems, die auf Ein/Ausgabe-Routinen zeigen, werden voreingestellt, d.h. sie erhalten die Inhalte, die ihnen auch in der Einschaltphase des Rechners gegeben werden.

Beispiel:

```
JSR     RESTOR
```

Funktionsname: SAVE

Einsprungadresse: $FFD8

Uebergaberegister: X, Y

Vorbereitungsroutinen: SETLFS, SETNAM, [MEMBOT]

Fehleranzeigen: 5, 8, 9

Stapelbedarf: 9

Beschreibung: Der Inhalt des Speichers ab der durch MEMBOT oder die Voreinstellung eingestellten Anfangsadresse bis zu der in X und Y uebergebenen Adresse wird an ein Ausgabegeraet uebertragen. Fuer das Kassettengeraet (Geraet 1) ist kein Filename erforderlich. Bei allen anderen Geraeten muss er angegeben werden, sonst wird eine Fehleranzeige gesetzt. Geraete 0 (Tastatur) und 3 (Bildschirm) sind fuer SAVE nicht definiert.

Beispiel:

```
LDX     TXTTAB
LDY     TXTTAB+1
JSR     MEMBOT
LDA     #1      ;KASSETTE
JSR     SETLFS
LDA     #0      ;KEIN FILENAME
LDX     VARTAB
LDY     VARTAB+1
JSR     SAVE
```

Funktionsname: SCNKEY

Einsprungadresse: $FF9F

Uebergaberegister: Keine

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 4

Beschreibung: Die Tastatur wird auf eine gedrueckte Taste abgefragt, deren ASCII-Code dann im Tastaturpuffer abgelegt wird. Diese Routine wird waehrend des Systeminterrupts, der alle 15 ms ausgeloest wird, durchlaufen.

Beispiel:

```
GET     JSR     SCNKEY ;TASTATUR ABFRAGEN
        JSR     GETIN  ;ZEICHEN HOLEN
        CMP     #0     ;KEIN ZEICHEN ?
        BEQ     GET    ;JA: WEITER FRAGEN
        JSR     CHROUT ;ZEICHEN ANZEIGEN
```

Funktionsname: SCREEN

Einsprungadresse: $FFED

Uebergaberegister: X, Y

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Die fuer den Rechner angegebene Spalten-Zeilen-Organisation wird im X- und Y-Register uebergeben. Beim VC20 also im X-Register 22 und im Y-Register 23.

Beispiel:

```
JSR     SCREEN
STX     MAXSPL
STY     MAXZEI
```

Funktionsname: SECOND

Einsprungadresse: $FF93

Uebergaberegister: A

Vorbereitungsroutinen: LISTEN

Fehleranzeigen: Siehe READST

Stapelbedarf: Keiner

Beschreibung: Eine Sekundaeradresse im Akkumulator wird nach Ausgabe eines LISTEN-Befehls an ein IEEE-Geraet ausgegeben. SECOND darf nicht zum Ausgeben einer Sekundaeradresse nach einem TALK-Befehl verwendet werden.

Beispiel:

```
;GERAET 8 MIT SEK.-ADR. 15
        LDA     #8
        JSR     LISTEN
        LDA     #15
        JSR     SECOND
```

Funktionsname: SETLFS

Einsprungadresse: $FFBA

Uebergaberegister: A, X, Y

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Logische Filenummer, Geraeteadresse und Sekundaeradresse werden gesetzt. Die logische Filenummer dient dem System dazu, Daten aus einer durch OPEN aufgebauten Tabelle zu lesen. Die Geraeteadresse darf zwischen 0 und 30 liegen. VC20 kennt folgende Geraeteadressen:

0 Tastatur
1 Kassettenstation
2 RS-232-Schnittstelle
3 Bildschirm
4 Drucker an seriellem Bus
8 Floppy Disk an seriellem Bus

Beispiel:

```
LDA     #32 ;LOG. FILENUMMER
LDX     #4  ;GERAET 4 (DRUCKER)
LDY     #255 ;KEINE SEK.-ADR.
JSR     SETLFS
```

Funktionsname: SETMSG

Einsprungadresse: $FF90

Uebergaberegister: A

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Dieses Unterprogramm steuert die Ausgabe von Fehler und Diagnosemeldungen des Betriebssystems. Sie wird mit einem Wert im Akkumulator aufgerufen. Die Bits 6 und 7 dieses Wertes steuern die Ausgabe der Meldung. Wenn Bit 7 gesetzt ist, handelt es sich um eine Fehlermeldung wie z.B:

DEVICE NOT PRESENT ERROR

Ist Bit 6 gesetzt, so wird eine Kontrollmeldung wie z.B:

PRESS PLAY ON TAPE

ausgegeben.

Beispiel:

```
        LDA     #$40  ;DIAGN.-MELDUNG
        JSR     SETMSG

        LDA     #0 ;ALLE SYSTEMMELDUNGEN
;                      UNTERDRUECKEN
        JSR     SETMSG
```

Funktionsname: SETNAM

Einsprungadresse: $FFBD

Uebergaberegister: A, X, Y

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Die Filenameninformation wird abgesetzt. Soll ein File ohne Namen eroeffnet werden, so muss seine Laenge mit 0 angegeben werden. Im Akkumulator wird die Namenlaenge und im X- und Y-Register der nieder- bzw. hoeherwertige Adressteil der Namensadresse angegeben. Die Namensadresse kann jede gueltige Speicheradresse sein, bei der der den Namen repraesentierende Zeichenstring abgelegt ist.

Beispiel:

```
        LDA     #NAME2-NAME
        LDX     #<NAME
        LDY     #>NAME
        JSR     SETNAM
        .
        .
        .
NAME    .BYTE   'FILE NAME'
NAME2
```

Funktionsname: SETTIM

Einsprungadresse: $FFDB

Uebergaberegister: A, X, Y

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Die Systemuhr wird gesetzt. Die im VC20 integrierte Systemuhr besteht aus einem 24-Bit-Zaehler, der durch den System-Interrupt alle 1/60 s (ca. 15 ms) fortgeschrieben wird. Der Zaehler zaehlt bis 5184000, was 24 Zeitstunden entspricht, und beginnt dann wieder bei 0. Um den Zaehler zu setzen, wird der Akkumulator mit den hoechstwertigen, das X-register mit den mittelwertigen und das Y-Register mit den niederwertigen 8 Bits des Zeitwertes gesetzt.

Beispiel:

```
;UHR AUF 10 MINUTEN = 36000 EINHEITEN
;SETZEN
        LDA     #0
        LDX     #>36000
        LDY     #<36000
        JSR     SETTIM
```

Funktionsname: SETTMO

Einsprungadresse: $FFA2

Uebergaberegister: A

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Zeitablauf bei IEEE-Verkehr setzen. Wenn der Akkumulator in Bit 7 eine 0 enthaelt, ist die Zeitablaufpruefung eingeschaltet und bei 1 ausgeschaltet. Wenn die Zeitablaufpruefung eingeschaltet ist, so muss ein IEEE-Geraet auf ein DAV-Signal innerhalb von 64 ms antworten. Andernfalls wird die Handshake-Folge abgebrochen. Fuer den Verkehr mit langsamen IEEE-Geraeten sollte diese Pruefung deshalb abgeschaltet werden.

Beispiel:

```
;ZEITABLAUFPRUEFUNG ABSCHALTEN
        LDA     #0
        JSR     SETTMO
```

Funktionsname: STOP

Einsprungadresse: $FFE1

Uebergaberegister: A

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 1

Beschreibung: Die STOP-Taste wird abgefragt. Wurde sie gedrueckt, so wird das Zero-Flip-Flop gesetzt und der Akkumulator ist 0. Alle anderen Flip-Flops bleiben unberuehrt. War die STOP-Taste nicht gedrueckt, so enthaelt der Akkumulator den Index einer Taste aus der untersten Tastatur-Reihe.

Beispiel:

```
JSR     STOP
BNE     *+5  ;STOP NICHT GEDRUECKT
JMP     READY ;STOP GEDRUECKT
.
.
.
```

Funktionsname: TALK

Einsprungadresse: $FFB4

Uebergaberegister: A

Vorbereitungsroutinen: Keine

Fehleranzeigen: Siehe READST

Stapelbedarf: 1

Beschreibung: Ein TALK-Befehl wird zusammen mit dem ATN-Signal an ein IEEE-Geraet gesendet, wobei vorher der Akkumulator mit dessen Adresse zwischen 0 und 30 geladen werden muss.

Beispiel:

```
;TALK-BEFEHL AN IEEE-GERAET NR. 8
        LDA     #8
        JSR     TALK
```

Funktionsname: TKSA

Einsprungadresse: $FF96

Uebergaberegister: A

Vorbereitungsroutinen: TALK

Fehleranzeigen: Siehe READST

Stapelbedarf: Keiner

Beschreibung: Eine Sekundaeradresse zwischen 0 und 31 im Akkumulator wird nach Aufruf des TALK-Unterprogramms an ein IEEE-Geraet gesendet. TKSA darf nicht zum Ausgeben einer Sekundaeradresse nach einem LISTEN-Befehl verwendet werden.

Beispiel:

```
;GERAET 4 MIT SEK.-ADR. 5
        LDA     #4
        JSR     TALK
        LDA     #5
        JSR     TKSA
```

Funktionsname:	UDTIM
Einsprungadresse:	$FFEA
Uebergaberegister:	Keine
Vorbereitungsroutinen:	Keine
Fehleranzeigen:	Keine
Stapelbedarf:	Keiner
Beschreibung:	Normalerweise wird dieses Unterprogramm vom Interrupt-Programm des Systems jede 1/60 Sekunde aufgerufen, um die Systemuhr fortzuschreiben und den Code fuer die ggfs. gedrueckte STOP-Taste zu speichern. Wenn Sie Ihre eigenen Interrupt-Programme schreiben, sollten Sie diese Routine mit beruecksichtigen, damit Systemuhr und STOP-Taste funktionsfaehig bleiben.

Funktionsname:	UNLSN
Einsprungadresse:	$FFAE
Uebergaberegister:	Keine
Vorbereitungsroutinen:	Keine
Fehleranzeigen:	Siehe READST
Stapelbedarf:	1
Beschreibung:	Ein UNLISTEN-Befehl wird ueber den IEEE-Bus ausgegeben.
Beispiel:	JSR UNLSN

Funktionsname:	UNTLK
Einsprungadresse:	$FFAB
Uebergaberegister:	Keine
Vorbereitungsroutinen:	Keine
Fehleranzeigen:	Siehe READST
Stapelbedarf:	1
Beschreibung:	Ein UNTALK-Befehl wird ueber den IEEE-Bus ausgegeben.

Beispiel:

```
JSR     UNTLK
```

Funktionsname: VECTOR

Einsprungadresse: $FF8D

Uebergaberegister: X, Y

Vorbereitungsroutinen: Keine

Fehleranzeigen: Keine

Stapelbedarf: 2

Beschreibung: Wird dieses Unterprogramm mit gesetztem Carry-Bit aufgerufen, so werden die Inhalte der Adresszeiger aus dem System-RAM in eine Liste uebertragen, deren Anfangsadresse im X- und Y-Register uebergeben werden muss. Bei geloeschtem Carry-Bit wird durch dieses Unterprogramm eine Anwender-Adresszeigerliste, deren Anfangsadresse im X-und Y-Register uebergeben werden muss, in die Adresszeiger des System-RAM uebertragen. Hierbei sollte vorsichtig vorgegangen werden. Es sollten zunaechst die Adresszeiger in die Anwendertabelle uebertragen, dort geaendert und anschliessend zurueckkopiert werden.

Beispiel:

```
;AENDERUNG DER ZEIGER FUER DIE EINGABE-
;ROUTINEN FUER EIN NEUES SYSTEM
        LDX     #<TABEL
        LDY     #>TABEL
        SEC
        JSR     VECTOR  ;ALTE ZEIGER LESEN
        LDA     #<EGNEU ;AENDERN
        STA     TABEL+10
        LDA     #>EGNEU
        STA     TABEL+11
        LDX     #<TABEL
        LDY     #>TABEL
        CLC
        JSR     VECTOR  ;SYSTEM AENDERN
        .
        .
        .
TABEL   *=*+26
```

Umsetzen von fremden Programmen in VC20-BASIC

C.1 Uebersicht

Wenn Sie ein Programm, das in der BASIC-Programmiersprache eines anderen Rechner-Systems geschrieben ist, auf Ihrem VC20 laufen lassen wollen, so sind einige geringfuegige Aenderungen erforderlich. Einige der wichtigsten Unterschiede zwischen der VC20-Version des CBM-BASIC und anderen BASIC-Interpretern werden in den folgenden Abschnitten beschrieben und es wird ein Weg angegeben, wie Sie solche Programme auf Ihrem VC20 lauffaehig machen koennen.

C.2 String-Dimensionen:

Loeschen Sie in dem fremden Programm Anweisungen, in denen Stringlaengen deklariert werden. Eine Anweisung, wie z.B:

```
DIM A$(I,J)
```

dimensioniert in manchen BASIC-Versionen ein eindimensionales Stringfeld mit J Strings der Laenge I. Ersetzen Sie dies durch:

```
DIM A$(J)
```

Manche BASIC-Versionen verwenden zur Kennzeichnung der String-Verkettung ein Komma (,) oder ein kaufmaennisches "und" (&). Solche Zeichen muessen Sie durch ein Pluszeichen (+) ersetzen.
Im VC20-BASIC werden zur Bildung von Teilstrings aus Strings die Funkti-onen:

```
MID$
RIGHT$
LEFT$
```

verwendet. Andere BASIC-Versionen verwenden z.B:

```
A$(I)
```

um das I-te Zeichen aus A$ zu extrahieren, oder:

```
A$(I,J)
```

um einen Teilstring von A$ zu bilden, der bei Position I beginnt und J Zeichen lang ist. In solchen Faellen schreiben Sie z.B:

fremdes BASIC	VC20-BASIC
A$(I)=X$	A$=LEFT$(A$,I-1)+X$+MID$(A$,I+1)
A$(I,J)=X$	A$=LEFT$(A$,I-1)+X$+MID$(A$,J+1)

C.3 Mehrfache Zuweisung

Um z.B. den Variablen B und C denselben Wert zuzuweisen, erlauben manche BASIC-Versionen:

```
10 LET B=C=0
```

VC20-BASIC wuerde in diesem Fall das zweite Gleichheitszeichen als logischen Operator auffassen und B auf -1 setzen, falls C=0. Deshalb muessen Sie in einem solchen Fall:

```
10 B=0:C=0
```

schreiben.

C.4 Verkettung von Anweisungen

Verschiedene BASIC-Versionen verwenden den nach links geneigten Schraegstrich (\), um mehrere Anweisungen in einer Zeile voneinander zu trennen. VC20-BASIC verwendet hier den Doppelpunkt.

C.5 MAT-Funktionen

Programme, die die in manchen BASIC-Versionen vorhandenen MAT-Funktionen verwenden, muessen mit Hilfe von FOR...NEXT-Schleifen umgeschrieben werden.

C.6 VC20-BASIC-Codes

Um Speicherplatz zu sparen, werden beim Eingeben von BASIC-Programmzeilen alle BASIC-Schluesselwoerter in 1-Byte-Codes verschluesselt und im Speicher abgelegt. Diese Codes finden Sie in der Tabelle C.1 auf der naechsten Seite.

Tabelle C.1: Codes fuer VC20-BASIC-Schluesselwoerter

Code (dez.)	BASIC-Wort	Code (dez.)	BASIC-Wort
128	END	167	THEN
129	FOR	168	NOT
130	NEXT	169	STEP
131	DATA	170	+
132	INPUT#	171	-
133	INPUT	172	*
134	DIM	173	/
135	READ	174	↑
136	LET	175	AND
137	GOTO	176	OR
138	RUN	177	>
139	IF	178	=
140	RESTORE	179	<
141	GOSUB	180	SGN
142	RETURN	181	INT
143	REM	182	ABS
144	STOP	183	USR
145	ON	184	FRE
146	WAIT	185	POS
147	LOAD	186	SQR
148	SAVE	187	RND
149	VERIFY	188	LOG
150	DEF	189	EXP
151	POKE	190	COS
152	PRINT#	191	SIN
153	PRINT	192	TAN
154	CONT	193	ATN
155	LIST	194	PEEK
156	CLR	195	LEN
157	CMD	196	STR$
158	SYS	197	VAL
159	OPEN	198	ASC
160	CLOSE	199	CHR$
161	GET	200	LEFT$
162	NEW	201	RIGHT$
163	TAB(	202	MID$
164	TO	203	GO
165	FN	204	?SYNTAX ERROR *)
166	SPC(		

*) Alle Codes zwischen 204 und 255 erzeugen diese Fehlermeldung, wenn sie mit LIST ausgelistet werden.

Zusammenstellung der Fehlermeldungen

D.1 Uebersicht

Dieser Anhang enthaelt eine tabellarische Zusammenstellung aller Fehler- und Diagnosemeldungen des VC20-BASIC-Interpreters und -Betriebssystems sowie eine detaillierte Beschreibung der Bedeutung jeder einzelnen Meldung.

Tabelle D.1: Fehlermeldungen

BASIC-Interpreter	Betriebssystem
BAD SUBSCRIPT	DEVICE NOT PRESENT
CAN'T CONTINUE	FILE NOT FOUND
DIVISION BY ZERO	FILE NOT OPEN
FILE DATA	FILE OPEN
FORMULA TOO COMPLEX	LOAD
ILLEGAL DIRECT	NOT INPUT FILE
ILLEGAL QUANTITY	NOT OUTPUT FILE
NEXT WITHOUT FOR	TOO MANY FILES
OUT OF DATA	VERIFY
OUT OF MEMORY	
OVERFLOW	
REDIM'D ARRAY	
REDO FROM START	
RETURN WITHOUT GOSUB	
STRING TOO LONG	
SYNTAX	
TYPE MISMATCH	
UNDEF'D FUNCTION	
UNDEF'D STATEMENT	

D.2 Fortsetzung des Programms nach einer Fehlermeldung

Nachdem ein Programm von einer Fehlermeldung abgebrochen wurde, kann es nicht mit einer im Direkt-Modus eingegebenen CONT-Anweisung fortgesetzt werden. Alle Variablen behalten jedoch ihre Werte, was bei der Fehlersuche hilfreich ist. GOSUB- und FOR...NEXT-Eintraege im Stapelspeicher werden durch die Unterbrechung geloescht, so dass eine Programmfortsetzung mit RETURN oder NEXT ebenfalls nicht moeglich ist.
In einem solchen Fall kann das Programm nur mit einer GOTO <Zeilennummer>-oder der RUN-Anweisung fortgesetzt bzw. neu gestartet werden.

D.3 Interpreter-Meldungen und ihre Bedeutung

BAD SUBSCRIPT

Es wurde eine indizierte Variable verwendet, deren Index groesser ist als der maximale in der DIM-Anweisung angegebene oder deren Index groesser als 10 ist, falls die Variable mit DIM nicht dimensioniert wurde.

CAN'T CONTINUE

Nach einem Programmabbruch durch eine Fehlermeldung oder nach einer Programmaenderung kann das Programm nicht durch CONT fortgesetzt werden.

DIVISION BY ZERO

Bei der Ermittlung eines mathematischen Ausdrucks ist der Nenner eines Bruches 0 geworden.

FILE DATA

Einer numerischen Variablen (Integer oder Gleitkomma) wurden bei einer INPUT#- oder GET#-Anweisung nichtnumerische Daten aus einem File zugewiesen.

FORMULA TOO COMPLEX

An einem Stringausdruck sind zu viele Teilstrings beteiligt. In diesem Fall muessen Sie den Ausdruck in mehrere Teilausdruecke zerlegen und die Zwischenergebnisse Stringvariablen zuweisen.

ILLEGAL DIRECT

GET, INPUT und DEF FN duerfen nicht im Direkt-Modus verwendet werden, da diese Anweisungen den Eingabepuffer benutzen, der jedoch zur Auswertung der Direkt-Anweisungen vom Interpreter benoetigt wird.

ILLEGAL QUANTITY

Es wurde eine Variable oder eine Funktion mit einem unerlaubten Wert oder Parameter verwendet. Diese Meldung wird in folgenden Faellen ausgegeben:

1. Ein Feldindex < 0 oder > 32767 wurde definiert.

2. LOG wurde mit negativem oder Null-Argument aufgerufen.

3. SQR wurde mit negativem Argument aufgerufen.

4. Es wurde ein mathematischer Ausdruck wie (-5)↑2.3 definiert.

5. Aufruf von USR, ehe die Startadresse des Maschinenspracheunterprogramms gespeichert wurde.

6. Verwendung der Stringfunktionen MID$, LEFT$ oder RIGHT$ mit ungueltigen Laengenparametern X (0<X<256).

7. Bei ON...GOTO oder ON...GOSUB wurde ein ungueltiger Index ermittelt.

8. Bei PEEK, POKE, WAIT oder SYS wurde eine ungueltige Adresse X spezifiziert (0<=X<65536).

9. Bei WAIT, POKE, TAB oder SPC wurden ungueltige Byte-Parameter X spezifiziert (0<=X<256).

NEXT WITHOUT FOR

Entweder wurden mehrere FOR...NEXT-Schleifen falsch geschachtelt oder zu einer NEXT-Anweisung fehlt eine vorausgegangene FOR-Anweisung.

OUT OF DATA

Eine READ-Anweisung versucht, mehr Daten aus einer DATA-Anweisung zu lesen, als vorhanden sind.

OUT OF MEMORY

Der Programmspeicher oder aber der Stapelspeicher sind voll. Im ersten Fall kann bei sehr grossen Programmen durch die Programmausfuehrung der verbleibende freie Speicher durch die Variablen, die waehrend der Programmabarbeitung abgelegt werden, ueberlaufen. Der zweite Fall kann durch zu viele geschachtelte FOR...NEXT-Schleifen, GOSUB...RETURN-Kombinationen oder durch zu viele Klammerebenen ausgeloest werden.

OVERLOW

Das Ergebnis einer Berechnung hat die im VC20 maximal darstellbare Zahl (1.70141183E+38) ueberschritten.

REDIM'D ARRAY

Es wurde versucht ein Feld mit gleichem Namen ein zweites Mal zu dimensionieren. Dieser Fehler tritt auch auf, wenn nach einer automatischen Dimensionierung mit einem Index < 11 das Feld mit einer DIM-Anweisung dimensioniert werden soll.

REDO FROM START

Dies ist keine Fehlermeldung im eigentlichen Sinne. Sie wird ausgegeben, wenn bei einer INPUT-Anweisung nichtnumerische Daten eingegeben wurden, wenn numerische erwartet wurden. Die Eingabe kann nach dieser Meldung wiederholt werden.

RETURN WITHOUT GOSUB

Es soll eine RETURN-Anweisung ausgefuehrt werden, der keine GOSUB-Anweisung vorausgegangen war.

STRING TOO LONG

Durch eine Stringverkettung ist ein String laenger als 255 Zeichen geworden oder es wurde versucht, mit INPUT# einen String von mehr als 80 Zeichen einzulesen.

SYNTAX

Der Interpreter findet in einer Befehlszeile eine Zeichenkombination, die er nicht versteht.

TYPE MISMATCH

Es wurde versucht, einem Variablentyp einen falschen Datentyp zuzuweisen (z.B. A$=B%) oder eine Funktion wurde mit einem falschen Argument versorgt (z.B. A=LEN(X%)).

UNDEF'D FUNCTION

Es wurde eine vom Anwender definierte Funktion aufgerufen, deren Definition (DEF FN) nicht im Programm existiert.

UNDEF'D STATEMENT

Es wurde versucht, mit GOTO, GOSUB oder THEN zu einer Zeilennummer zu verzweigen, die nicht im Programm existiert.

D.4 Betriebssystemfehlermeldungen und ihre Bedeutung

DEVICE NOT PRESENT

Es wurde versucht, ein Geraet am seriellen System-Bus oder am IEEE-Bus zu adressieren (durch OPEN, CLOSE, CMD, INPUT#, GET# oder PRINT#), wenn entweder kein Geraet angeschlossen oder kein Geraet angeschaltet ist.

FILE NOT FOUND

Der in einer OPEN- oder LOAD-Anweisung spezifizierte File konnte auf dem spezifizierten Eingabegeraet nicht gefunden werden. Bei der Kassette wurde eine Band-Ende-Marke erkannt.

FILE NOT OPEN

Es wurde mit einer INPUT#-, GET#- oder PRINT#-Anweisung eine logische Filenummer angesprochen, der durch eine entsprechende OPEN-Anweisung noch kein Geraet zugeordnet wurde.

FILE OPEN

Es wurde versucht, eine zweite OPEN-Anweisung mit derselben logischen Filenummer ohne dazwischen liegende CLOSE-Anweisung mit dieser Filenummer auszufuehren.

LOAD

Diese Meldung wird ausgegeben, wenn beim Laden eines Programms von Kassette im Originalblock mehr als 31 Fehler, oder wenn im Originalblock und in der Kopie Fehler an derselben Stelle auftreten.

NOT INPUT FILE

Es wurde versucht, aus einem File, der zum Schreiben eroeffnet wurde, Daten zu lesen.

NOT OUTPUT FILE

Es wurde versucht, in einen File, der zum Lesen eroeffnet wurde, Daten zu schreiben.

TOO MANY FILES

Es koennen gleichzeitig nur bis zu 10 eroeffnete logische Files vom Betriebssystem verwaltet werden. Wird versucht, mit OPEN einen weiteren File zu eroeffnen, so wird diese Fehlermeldung ausgegeben.

VERIFY

Beim Vergleich zwischen dem Inhalt eines Speicherbereiches (Programm) und dem Inhalt des korrespondierenden Files auf einem peripheren Geraet tritt Ungleichheit auf.

Mathematische Funktionen und ASCII-Codes

E.1 Trigonometrische, zyklometrische und Hyperbel-Funktionen

Einige der trigonometrischen, zyklometrischen und alle Hyperbelfunktionen sind im VC20-BASIC nicht implementiert. Sie lassen sich jedoch durch die vorhandenen Funktionen ersetzen, wie die unten stehende Aufstellung zeigt.

Funktion(X)	VC20-BASIC-Aequivalent-Funktion(X)
SECANS	1/COS(X)
COSECANS	1/SIN(X)
COTANGENS	1/TAN(X)
ARCUS SINUS	ATN(X/SQR(-X*X+1))
ARCUS COSINUS	-ATN(X/SQR(-X*X+1))+<pi>/2
ARCUS SECANS	ATN(X/SQR(X*X-1))
ARCUS COSECANS	ATN(X/SQR(X*X-1))+SGN(X)-1)*<pi>/2
ARCUS COTANGENS	ATN(X)+<pi>/2
SINUS HYPERBOLICUS	(EXP(X)-EXP(-X))/2
COSINUS HYPERBOLICUS	(EXP(X)+EXP(-X))/2
TANGENS HYPERBOLICUS	(EXP(X)-EXP(-X))/(EXP(X)+EXP(-X))
SECANS HYPERBOLICUS	2/(EXP(X)+EXP(-X))
COSECANS HYPERBOLICUS	2/(EXP(X)-EXP(-X))
COTANGENS HYPERBOLICUS	(EXP(X)+EXP(-X))/(EXP(X)-EXP(-X))
AREA SINUS HYPERBOLICUS	LOG(X+SQR(X*X+1))
AREA COSINUS HYPERBOLICUS	LOG(X+SQR(X*X-1))
AREA TANGENS HYPERBOLICUS	LOG((1+X)/(1-X))/2
AREA SECANS HYPERBOLICUS	LOG((SQR(-X*X+1)+1)/X)
AREA COSECANS HYPERBOLICUS	LOG((SGN(X)*SQR(X*X+1)+1)/X)
AREA COTANGENS HYPERBOLICUS	LOG((X+1)/(X-1))/2

E.2 ASCII-Zeichencode

Die Aufstellung auf der naechsten Seite enthaelt den CBM-modifizierten ASCII-Zeichencode in dezimaler und hexadezimaler Schreibweise. Der VC20-Zeichensatz weicht insofern von diesem Standard ab, als im Gross/Grafik-Modus anstelle der Kleinbuchstaben Grafiksymbole codiert sind. Im Gross-/Kleinschreib-Modus sind die Kleinbuchstaben anstelle der Grossbuchstaben codiert waehrend den Grossbuchstaben ein um 128 gegenueber den Kleinbuchstaben erhoehter Code zugeordnet ist.

ASCII dez	ASCII hex	Zeichen	ASCII dez	ASCII hex	Zeichen	ASCII dez	ASCII hex	Zeichen
000	$00	NULL	043	$2B	+	086	$56	V
001	$01	SOH	044	$2C	,	087	$57	W
002	$02	STX	045	$2D	-	088	$58	X
003	$03	ETX	046	$2E	.	089	$59	Y
004	$04	EOT	047	$2F	/	090	$5A	Z
005	$05	ENQ	048	$30	0	091	$5B	[
006	$06	ACK	049	$31	1	092	$5C	\
007	$07	BEL	050	$32	2	093	$5D	]
008	$08	BS	051	$33	3	094	$5E	↑
009	$09	HT	052	$34	4	095	$5F	Pfeil links
010	$0A	LF	053	$35	5	096	$60	Leerstelle
011	$0B	VT	054	$36	6	097	$61	a
012	$0C	FF	055	$37	7	098	$62	b
013	$0D	CR	056	$38	8	099	$63	c
014	$0E	SO	057	$39	9	100	$64	d
015	$0F	SI	058	$3A	:	101	$65	e
016	$10	DLE	059	$3B	;	102	$66	f
017	$11	DC1	060	$3C	<	103	$67	g
018	$12	DC2	061	$3D	=	104	$68	h
019	$13	DC3	062	$3E	>	105	$69	i
020	$14	DC4	063	$3F	?	106	$6A	j
021	$15	NAK	064	$40	§	107	$6B	k
022	$16	SYN	065	$41	A	108	$6C	l
023	$17	ETB	066	$42	B	109	$6D	m
024	$18	CAN	067	$43	C	110	$6E	n
025	$19	EM	068	$44	D	111	$6F	o
026	$1A	SUB	069	$45	E	112	$70	p
027	$1B	ESC	070	$46	F	113	$71	q
028	$1C	FS	071	$47	G	114	$72	r
029	$1D	GS	072	$48	H	115	$73	s
030	$1E	RS	073	$49	I	116	$74	t
031	$1F	US	074	$4A	J	117	$75	u
032	$20	Leerstelle	075	$4B	K	118	$76	v
033	$21	!	076	$4C	L	119	$77	w
034	$22	"	077	$4D	M	120	$78	x
035	$23	#	078	$4E	N	121	$79	y
036	$24	$	079	$4F	O	122	$7A	z
037	$25	%	080	$50	P	123	$7B	;
038	$26	&	081	$51	Q	124	$7C	<
039	$27	'	082	$52	R	125	$7D	=
040	$28	(	083	$53	S	126	$7E	>
041	$29	)	084	$54	T	127	$7F	?
043	$30	*	085	$55	U			

Das 6561-Video-Interface-Chip (VIC)

F.1 Allgemeine Beschreibung

Der VIC-Baustein ist nach dem Mikroprozessor 6502 das vielseitigste Bauelement des VC20. Er wurde fuer Farbvideo-Bildschirme und Video-Heimspiele entwickelt und enthaelt als integrierter Baustein alle Schaltkreise, die zur Erzeugung farb-programmierbarer Zeichengrafik mit hoher Bildschirmaufloesung erforderlich sind. Ausserdem enthaelt VIC Tongeneratoren fuer akustische Effekte sowie Analog-Digital-Wandler fuer Video-Spiele.

F.2 Eigenschaften

Im einzelnen hat der VIC-Baustein folgende Eigenschaften:

* Voll ausbaubares System mit 16-kByte-Adressraum.
* Verwendung industriekompatibler 8-Bit-ROMs und 4-Bit-RAMs.
* Masken-programmierbare Synchronisation fuer PAL.
* Erzeugung von 16 verschiedenen Farbtoenen.
* Bis zu 600 unabhaengig voneinander programmierbare und verschiebbare Hintergrundplaetze auf einem Standardfernsehbildschirm.
* Bildschirm-Zentrierung.
* Schirmgittergroesse bis zu 192 Punkte horizontal und 200 Punkte vertikal.
* Zwei waehlbare grafische Zeichengroessen.
* Akustisches System, bestehend aus drei unabhaengig voneinander programmierbaren Tongeneratoren, einem Generator fuer weisses Rauschen sowie einem Amplitudenmodulator.
* 2 8-Bit-Analog-Digital-Wandler.
* DMA und Adress-Generierung.

* Keine CPU-Wartezeiten oder Bildschirmueberlagerungen waehrend der Bildwiederholung.

* Wahlschalter fuer Zwischenzeilenabtastung.

* 16 programmierbare Kontrollregister.

* Lichtkananone oder Lichtstift fuer Videospiele.

* 2 Farb-Betriebsmodi.

F.3 Arbeitsweise

Zur Erzeugung programmierbarer farbiger Zeichen greift VIC auf externe Speicher zu, die in drei Bereiche unterteilt werden koennen:

Zeichenzeiger
anzuzeigende Zeichen
Farbzeiger

Der Bereich der Zeichenzeiger ist ein RAM-Bereich mit 506 Speicherplaetzen, der als Video-Matrix bezeichnet wird. Jedem dieser Speicherplaetze entspricht eine Zeichenposition auf dem Bildschirm.
Der Bereich der anzuzeigenden Zeichen besteht aus einem Satz von 8- oder 16-Byte-Bloecken. Jeder dieser Bloecke enthaelt das Punktraster fuer ein einzelnes anzuzeigendes Zeichen. Die Bloecke koennen entweder im RAM- oder im ROM-Bereich untergebracht werden.
Der Farbzeigerbereich schliesslich besteht aus 506 Farbbytes oder -zellen, in denen nur die niederwertigen 4 Bits zur Definition der Farbe, in der das an der korrespondierenden Stelle stehende Zeichen abzubilden ist sowie zur Wahl einer der beiden Farb-Modi dienen. Dieser Bereich wird als Farb-Matrix bezeichnet.
Die Organisation der Video- und der Farb-Matrix sowie der Zeichenzellen uebernimmt dabei der Mikroprozessor.
Zum Verstaendnis der Arbeitsweise des VIC-Bausteins betrachten Sie jetzt bitte die Abbildung F.1 auf der naechsten Seite. Dies ist eine typische Video-Matrix, die das Abbild des Bildschirms mit 23 Zeilen zu 22 Spalten darstellt, also einen Bereich von 506 Anzeigeplaetzen bei einer Bildschirmaufloesung von 176 Punkten horizontal und 184 Punkten vertikal. Wenn jetzt eine Taste gedrueckt wird, so wird der zugehoerige Zeichenindex in die Video-Matrix uebertragen. Bei dem Beispiel in Abb. F.1 ist es der Index $2B, der an der Position $0B/$15 (Zeile/Spalte) in die Video-Matrix gespeichert wurde. VIC liest diesen Index aus der Matrix und fuehrt eine Adressberechnung durch, um das an dieser Position anzuzeigende Zeichen zu ermitteln. Diese Berechnung sieht fuer den Fall einer 8x8-Bit-Zeichenzelle folgendermassen aus:
Der Index wird dreimal linksgeshiftet (Multiplikation mit 8). Das Ergebnis wird zur Startadresse des Zeichenzellenbereiches ($8000 beim VC20), die in der Initialisierungsphase in das Kontrollregister CR5 des VIC gespeichert wird, addiert, woraus sich die Adresse $8158 in unserem Beispiel ergibt. Unter dieser Adresse findet VIC das anzuzeigende Punktraster in einem 8x8-Bit-Zeichenzellenbereich.

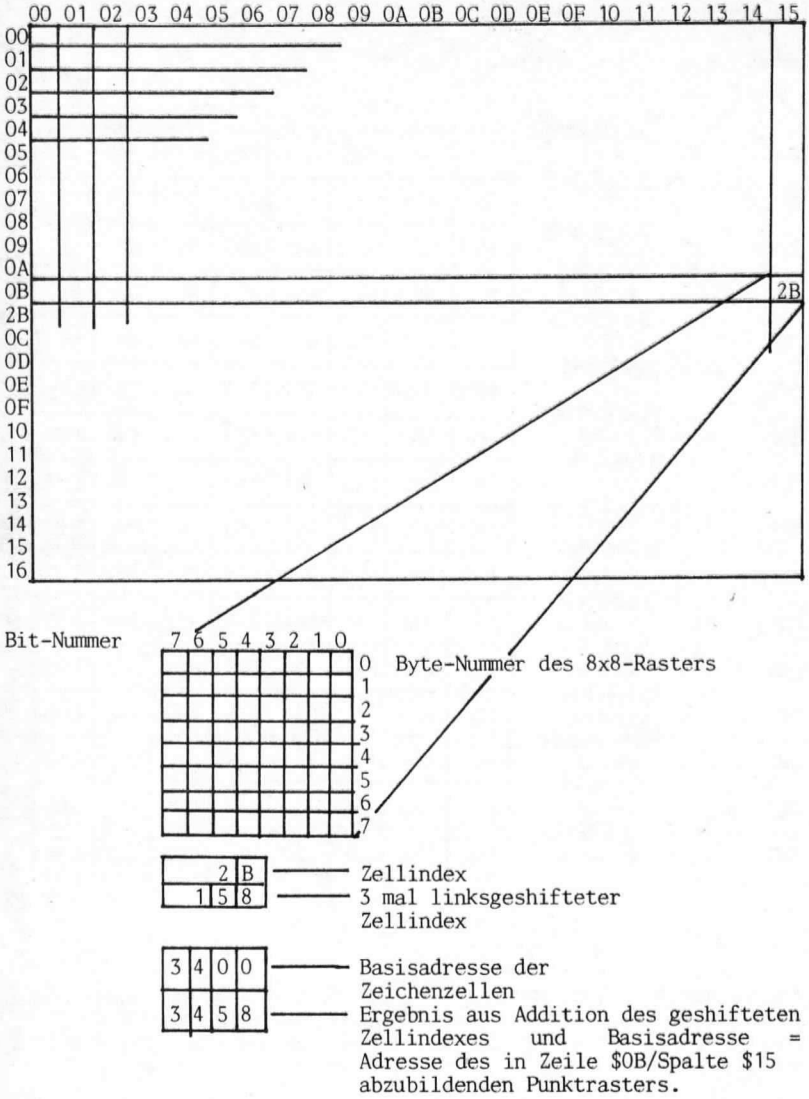


Abb. F.1: Typische Video-Matrix

F.4 Die VIC-Kontrollregister

Bezeichnung	Adresse (hex)	Bit 7	6	5	4	3	2	1	0
CR0	$9000	I	SX6	SX5	SX4	SX3	SX2	SX1	SX0
CR1	$9001	SY7	SY6	SY5	SY4	SY3	SY2	SY1	SY0
CR2	$9002	BV9	M6	M5	M4	M3	M2	M1	M0
CR3	$9003	R0	N5	N4	N3	N2	N1	N0	D
CR4	$9004	R8	R7	R6	R5	R4	R3	R2	R1
CR5	$9005	BV13	BV12	BV11	BV10	BC13	BC12	BC11	BC10
CR6	$9006	LH7	LH6	LH5	LH4	LH3	LH2	LH1	LH0
CR7	$9007	LV7	LV6	LV5	LV4	LV3	LV2	LV1	LV0
CR8	$9008	PX7	PX6	PX5	PX4	PX3	PX2	PX1	PX0
CR9	$9009	PY7	PY6	PY5	PY4	PY3	PY2	PY1	PY0
CRA	$900A	S1	F16	F15	F14	F13	F12	F11	F10
CRB	$900B	S2	F26	F25	F24	F23	F22	F21	F20
CRC	$900C	S3	F36	F35	F34	F33	F32	F31	F30
CRD	$900D	S4	F46	F45	F44	F43	F42	F41	F40
CRE	$900E	CA3	CA2	CA1	CA0	A3	A2	A1	A0
CRF	$900F	CB3	CB2	CB1	CB0	R	CE2	CE1	CE0

Abb. F.2: VIC-Kontrollregister

Die in Abb. F.2 dargestellten 16 8-Bit-Kontrollregister des VIC6561 gestatten es dem Mikroprozessor, alle Betriebsmodi des VIC zu steuern. Im folgenden werden die einzelnen Register und ihre Funktion detailliert beschrieben:

CR0: Die Bits 0 bis 6 bestimmen, in welcher Entfernung vom linken Rand des Bildschirms die erste Zeichenspalte beginnt. Damit koennen die verschiedenen Bildschirmgroessen horizontal zentriert werden. Mit Bit 7 wird Zwischenzeilenabtastung gewaehlt (I=1).

CR1: Die Bits 0 bis 7 bestimmen, in welcher Entfernung vom oberen Rand des Bildschirms die erste Zeichenzeile beginnt. Damit koennen die verschiedenen Bildschirmgroessen vertikal zentriert werden.

CR2: Die Bits 0 bis 6 setzen die Spaltenzahl der Video-Matrix. Bit 7 ist ein Bestandteil der in Register CR5 gespeicherten Anfangsadresse der Video-Matrix.

CR3: Die Bits 1 bis 6 setzen die Zeilenzahl der Video-Matrix. Bit 0 waehlt entweder 8x8-Bit-Zeichenmatrizen (D=0) oder 16x8-Bit-Zeichenmatrizen (D=1). Bit 7 ist Bestandteil des Rasterwertes aus Register CR4.

CR4: Enthaelt die Nummer der momentan vom Rasterstrahl abgetasteten Zeile.

CR5: Die Bits 0 bis 3 bestimmen die Startadresse des Zeichenzellenbereiches. Sie bilden die Adressbits A13 bis A10 der aktuellen Adresse. Die Bits 4 bis 7 zusammen mit Bit 7 aus CR2 bestimmen die Anfangsadresse der Video-Matrix. Sie bilden die Adressbits A13 bis A9 der aktuellen Adresse.

CR6: Enthaelt die zwischengespeicherte Horizontalposition der Lichtkanone oder des Lichtstiftes.

CR7: Enthaelt die zwischengespeicherte Vertikalposition der Lichtkanone oder des Lichtstiftes.

CR8: Enthaelt den digitalisierten Wert des X-Potentiometers.

CR9: Enthaelt den digitalisierten Wert des Y-Potentiometers.

CRA: Die Bits 0 bis 6 setzen die Frequenz des ersten Tongenerators. Bit 7 schaltet ihn ein (S1=1) oder aus (S1=0).

CRB: Wie CRA, nur fuer den zweiten Tongenerator.

CRC: Wie CRA, nur fuer den dritten Tongenerator.

CRD: Wie CRA, nur fuer den Generator fuer weisses Rauschen.

CRE: Die Bits 0 bis 3 setzen die Lautstaerke des zusammengesetzten Geraeuschsignals. Es muss mindestens ein Generator zur Geraeuscherzeugung eingeschaltet sein. Die Bits 4 bis 7 enthalten den in Verbindung mit dem Mehrfarbbetriebsmodus verwendeten Hilfs-Farbcode.

CRF: Die Bits 4 bis 7 waehlen einen von 16 Farbtoenen fuer die allen Zeichen gemeinsame Hintergrundfarbe. Sie setzen grundsaetzlich die Farbe fuer den Hintergrund innerhalb der Video-Matrix. Die Bits 0 bis 2 waehlen einen von 8 Farbtoenen fuer die Rahmenfarbe, also fuer den Bereich ausserhalb der Video-Matrix. Bit 3 bestimmt, ob verschiedenfarbige Zeichen auf einem einfarbigen Hintergrund (R=1) oder ob gleichfarbige Zeichen auf einem bei jedem Zeichen andersfarbigen Hintergrund (R=0) abgebildet werden. Das R-Bit hat keine Funktion, wenn Vielfarb-Modus gewaehlt wurde.

F.5 Ein Beispiel fuer die VIC-Kontrollregister-Anwendung

Zum noch besseren Verstaendnis des VIC-Bausteines wird im folgenden ein Beispiel fuer die Kontrollregisteranwendung des VIC-Bausteins gegeben.
Zur Vereinfachung sei angenommen, dass alle Zeichen im hochaufloesenden Modus dargestellt werden und dass die Kontrollregister mit den folgenden Werten geladen sind:

Register	Inhalt hex.	Inhalt bin.	Ergebnis
CR0	$03	0 0000011	Verschiebt den Ursprung der Video-Matrix um 3(x4) Punktbreiten nach rechts. Zwischenzeilenabtastung ist ausgeschaltet (I=0).
CR1	$19	00011001	Verschiebt den Ursprung der Video-Matrix um 25(x2) Punkthoehen vom oberen Bildschirmrand nach unten.
CR2	$96	1 0010110	Setzt 22 Videomatrixspalten. Bit 7 gehoert zu Register 5.
CR3	$2E	X 010111 0	Setzt 23 Videomatrixzeilen. Es sind 8x8-Bit-Zeichenmatrizen gewaehlt (D=0).

CR5 sollte fuer den Zugriff auf die geeigneten Speicherplaetze gesetzt werden. Nehmen wir an, die Videomatrix beginnt bei $0200 und der Zeichenzellenbereich bei $3400. Dann muss CR5 folgendermassen gesetzt werden:

CR5	$0D	0000 1101	Dazu gehoert noch Bit 7 von CR2.

Auf der naechsten Seite wird dargestellt, wie die Adressen aus dieser Information erzeugt werden.

Video-Matrix-Startadresse (14 Bits):

```
CR5-Bits      CR2-Bit
7 6  5 4 7 ----'

0 0  0 0 1 0  0 0 0 0  0 0 0 0   binaerer Inhalt

 0    2        0        0       hexadezimaler Inhalt
```

Zeichenzellenbereich-Startadresse (14 Bits):

```
CR5-Bits
3 2  1 0

1 1  0 1 0 0  0 0 0 0  0 0 0 0   binaerer Inhalt

 3    4        0        0       hexadezimaler Inhalt
```

Register	Inhalt (hex)	Inhalt (bin)	Ergebnis
CRA	$00	0 0000000	Tongenerator 1 ist ausgeschaltet.
CRB	$9A	1 0011010	Tongenerator 2 ist mit einer relativen Frequenz von 26 eingeschaltet.
CRC	$00	0 0000000	Tongenerator 3 ist ausgeschaltet.
CRD	$A5	1 0000000	Der Rauschgenerator ist mit einer relativen Frequenz von 37 eingeschaltet.
CRE	$XF	XXXX 1111	Die Geraeuscheffekte sind auf groesste Lautstaerke gestellt.
CRF	$0E	0000 1 110	Die allen Zeichen gemeinsame Hintergrundfarbe ist Schwarz (0). Die Rahmenfarbe ist dunkelblau (6) und jedes Zeichen wird in der ihm zugeordneten Farbe auf dem schwarzen Hintergrund abgebildet (R=1).

Zusammengefasst ergibt sich aus den so eingestellten Registern folgender Betriebszustand:
Es wird eine zentrierte Videomatrix mit 22 Spalten und 23 Zeilen erzeugt. Jedes Zeichen erscheint farbig auf einem schwarzen, dunkelblau eingerahmten Hintergrund. Dabei wird als Geraeuscheffekt eine mittlere Tonschwingung zusammen mit weissem Rauschen in groesster Lautstaerke erzeugt.

Alle beschriebenen Register koennen fuer verschiedene Effekte mit anderen Inhalten versehen werden. Wenn der Inhalt von CR0 erhoeht wird, verschiebt sich der Video-Matrix-Bereich weiter nach rechts. Wird der Inhalt von CRB verringert (Bit 7 bleibt 1), so verringert sich die Frequenz von Tongenerator 2. Wenn CRF auf $06 veraendert wird, indem Bit R geloescht wird, erscheinen schwarze Zeichen auf unterschiedlich farbigem Hintergrund (Invers-Modus) und der Rahmen bleibt dunkelblau.

F.5 Farb-Betriebs-Modi

Der VIC6561-Baustein erlaubt zwei verschiedene Farb-Betriebs-Modi, naemlich den Modus fuer hohe Bildschirmaufloesung und den Vielfarb-Modus. Grundsaetzlich bestimmt der jeweilige Modus, wie die Information aus den Zeichenzellen in Bildpunkte auf dem Bildschirm umgesetzt wird. Der Modus wird durch das hoechstwertige Bit des Farbzeigers bestimmt, der jedem Speicherplatz in der Video-Matrix zugeordnet ist. Ist dieses Bit eines Farbzeigers 0, so wird das betreffende Zeichen in hoher Aufloesung, andernfalls im Vielfarb-Modus angezeigt.
Beim Modus fuer hohe Bildschirmaufloesung besteht ein Eins-zu-Eins-Zusammenhang zwischen gesetzten Bits in den Zeichenmatrizen und den Bildschirmpunkten, d.h. alle 1-er-Bits eines Zeichens werden in einer Farbe und alle 0-er-Bits in einer anderen Farbe dargestellt. Die Vordergrundfarbe des Zeichens wird durch die restlichen 3 Bits des Zeichenfarbzeigers bestimmt, waehrend die Zeichenhintergrundfarbe durch das Register CRF bestimmt wird.
Beim Vielfarb-Modus sind immer 2 Bits einer Zeichenzelle einem Bildschirmpunkt zugeordnet, dessen Farbton durch den mit den 2 Bits darstellbaren Code bestimmt wird. Es sind also 4 Farben fuer jedes Zeichen moeglich. Da jedoch immer 2 Bits der Zellendaten zu einem Bildpunkt gehoeren, ist die Horizontalaufloesung nur noch halb so gross, d.h. jeder 8x8-Bit-Zeichenzelle im Speicher ist ein 8x4-Punkt-Zeichen auf dem Bildschirm zugeordnet. Die durch die 2 Bits erzeugten 4 verschiedenen Codes geben an, wo die Farbinformation fuer den jeweiligen Bildpunkt zu finden ist. Diese Farbe kann die Hintergrundfarbe (CRF), die Rahmenfarbe (CRF), die Hilfsfarbe (CRE) oder die Vordergrundfarbe (Bits 0 bis 2 des Zeichenfarbzeigers) sein. Die vier Codes sind:

00: Hintergrundfarbe (CRF)
01: Rahmenfarbe (CRF)
10: Vordergrundfarbe (Farbzeiger)
11: Hilfsfarbe (CRE)

Es ist zu beachten, dass dieser Code kein eigentlicher Farbcode ist, sondern ein Zeiger auf vier verschiedene Farbinformationen, die selbst 3- bzw. 4-Bit-Informationen sind.
Auf der naechsten Seite ist ein Beispiel angegeben, das die Arbeitsweise der beiden Farb-Betriebs-Modi verdeutlichen soll.

Gegeben sei:

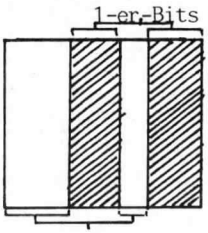
CRF: $1F — Farbhintergrund ist WEISS (1).
Rahmenfarbe ist GELB (7).
Keine inverse Darstellung (R=1).

CRE: $6X — Hilfsfarbe ist BLAU (6).

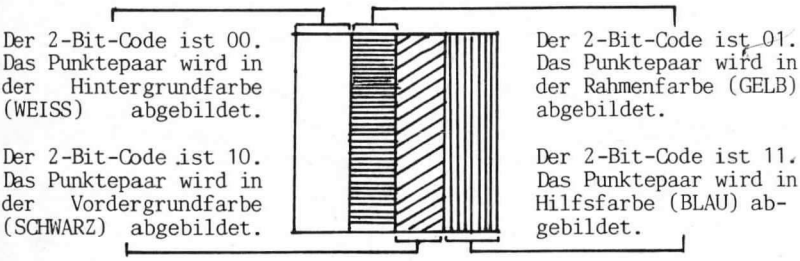
Zeichenraster:

	Bit	7	6	5	4	3	2	1	0	Hex-Code
Byte	0	0	0	0	1	1	0	1	1	$1B
	1	0	0	0	1	1	0	1	1	$1B
	2	0	0	0	1	1	0	1	1	$1B
	3	0	0	0	1	1	0	1	1	$1B
	4	0	0	0	1	1	0	1	1	$1B
	5	0	0	0	1	1	0	1	1	$1B
	6	0	0	0	1	1	0	1	1	$1B
	7	0	0	0	1	1	0	1	1	$1B

Wenn die Farbzelle fuer dieses Zeichen 0 enthaelt (0000), dann ist die Vordergrundfarbe SCHWARZ (0) und es ist der Modus fuer hohe Aufloesung gewaehlt (hoechstwertiges Bit ist Null. Das Zeichen wird dann folgendermassen abgebildet:



Wenn die Farbzelle fuer dieses Zeichen 8 enthaelt (1000), dann ist die Vordergrundfarbe SCHWARZ (0) und es ist der Vielfarb-Modus gewaehlt (hoechstwertiges Bit ist 1). Das Zeichen wird dann folgendermassen abgebildet:



F.7 VIC6561-Pin/Signal-Beschreibung

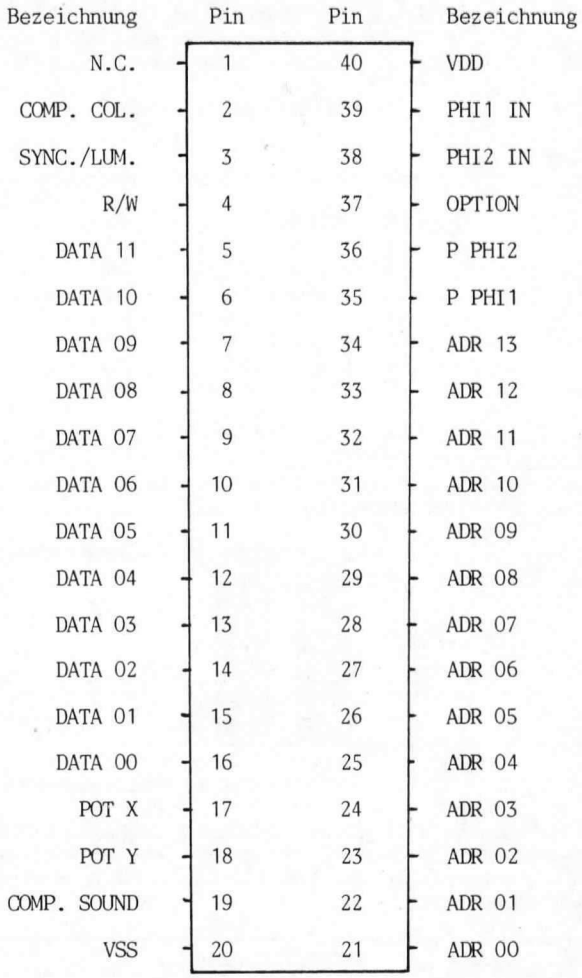


Abb. F.3: Pin-Belegung des VIC6561

Zum Abschluss dieses Anhangs werden auf den folgenden Seiten die einzelnen Pins gemaess obiger Abbildung beschrieben:

Adress-Bus (Pins 21 bis 34):

Der 14-Bit-Adress-Bus (ADR 00 bis ADR 13) ist bidirektional. Waehrend P PHI2=1 ist, sind die Adresspins auf Eingabemodus geschaltet. In diesem Modus hat der Mikroprozessor Zugriff auf jedes der 16 VIC-Kontrollregister. Die oberen 6 Pins des Adressbus' (ADR 08 bis ADR 13) wirken im Eingabemodus als Chip-Select-Pins. Eine "wahre" Chip-Select-Bedingung ist eingetreten, wenn:

ADR 13=ADR 11=ADR 09=ADR 08=0 und ADR 12=1,

was einer VIC-Chip-Select-Adresse von $1000 entspricht. Die unteren 4 Pins des Adressbus' (ADR 00 bis ADR 03) dienen als Kontrollregister-Select-Teil der Eingabeadresse. Waehrend P PHI1=1 ist, sind die Adresspins auf Ausgabemodus geschaltet, wenn Daten (Zeichen- oder Farbzeiger) gelesen werden. Die Adresse von VIC wird 50 ns nach der positiven Flanke von P PHI1 gueltig und bleibt es bis zur positiven Flanke von P PHI2.

Schreiben/Lesen (Pin 4):

Dies ist beim 6561 ein reines Eingangssignal und steuert den Datenfluss zwischen VIC und Mikroprozessor. Wenn das Schreib/Lese-Signal auf Masse liegt und die Chip-Select-Bedingungen erfuellt sind, kann der Mikroprozessor in das angewaehlte VIC-Kontrollregister schreiben. Liegt das Schreib/Lese-Signal dagegen hoch, so liest der Mikroprozessor aus dem angewaehlten VIC-Kontrollregister.
Beachten Sie, dass der Datenaustausch zwischen Prozessor und VIC nur bei P PHI2=1 moeglich ist. Bei P PHI1=1 liest VIC Daten fuer die Anzeige aus dem Speicher, wobei das Schreib-/Lese-Signal hochgehalten werden muss, um zu verhindern, dass VIC in irgend eine Speicherzelle schreibt.

Datenbus (Pins 5 bis 16):

Der 12-Bit-Datenbus des VIC6561 (DATA 00 bis DATA 11) ist in zwei Bereiche unterteilt. Die unteren 8 Bits (DATA 00 bis DATA 07) dienen sowohl dem Datenaustausch mit dem Mikroprozessor als auch zum Lesen von anzuzeigenden Daten, waehrend die oberen 4 Bits (DATA 08 bis DATA 11) ausschliesslich zur Uebernahme von Farb- und Modus-Informationen dienen.
Waehrend P PHI2=1 ist, werden ausschliesslich Daten zwischen Prozessor und VIC ueber DATA 00 bis DATA 07 uebertragen. Bei P PHI1=1 list VIC anzuzeigende Daten (Zeichen) ueber DATA 00 bis DATA 07.

Master-Oszillator-Takteingang (PHI 1 und PHI 2, Pins 39 und 38):

VIC6561 benoetigt einen 4,433618-MHz-Takt fuer den PAL-Standard. Die Taktsignale muessen 5 V betragen und duerfen sich nicht ueberlappen.

Systemtakt (P PHI1 und P PHI2, Pins 35 und 36):

Dies ist der Master-Systemtakt-Ausgang fuer den VC20 mit 5 V Signalspannung, 1,108 MHz und nichtueberlappend.

Speichertakt (PHIM (wahlweise), Pin 37):

Dies ist ein Einphasen-2,217-MHz-Takt, der nur dann erforderlich ist, wenn die Speicher des VIC-Systems ein Strobe-Signal benoetigen, nachdem der Adressbus gueltige Information enthaelt.

Analog-Digital-Wandler (POT X und POT Y, Pins 17 und 18):

Mit diesen Eingaengen werden Potentiometerstellungen in vom Mikroprozessor lesbare 8-Bit-Binaerzahlen durch einfache RC-Zeitkonstantenintegration umgewandelt. Eine externe an einen POT-Eingang angeschlossene Kapazitaet wird dabei ueber das Potentiometer aufgeladen.

Mischgeraeusche (Pin 19):

Dies ist der Ausgang des Ton-Synthesizers des VIC. Es ist ein hochohmiger Ausgang (ca. 1 kOhm), der fuer den Betrieb eines Lautsprechers extern gepuffert und verstaerkt werden muss.

Synchronisation und Helligkeit (Pin 3):

Dies ist ein Open-Collector-Ausgang, der die Information fuer die Synchronisation und Helligkeit eines Standard-Farbfernsehschirms liefert.

Farbe (Pin 2):

Der Farbausgang liefert die vollstaendige Information, die ein Farbfernsehschirm fuer den Empfang eines Farbbildes benoetigt. Es ist ein hochohmiger Ausgangspuffer, der sowohl das Referenz-Burst-Signal als auch die farb-codierte Phasen- und Amplitudeninformation bei 4,433618 MHz liefert.

Reset (wahlweise, Pin 37)

Der Reset-Ausgang gibt den Zustand des VIC bezueglich des Zugriffs auf den Videospeicher wieder. Der Ausgang geht 2 Mikrosekunden, bevor VIC auf den Speicher zugreift, auf Masse und bleibt so, bis das gesamte Bild wiederholt wurde.

Licht-Kanone/Licht-Stift (wahlweise, Pin 37):

Durch dieses Eingangssignal wird die gegenwaertig abgetastete Bildschirmposition eines Bildpunktes in die Kontrollregister CR6 und CR7 bei negativer Flanke zwischengespeichert. Dieser Eingang wird zusammen mit einer Photozelle bei Video-Spielen oder fuer einen Lichtstift verwendet.

Commodore GmbH
Lyoner Straße 38
D-6000 Frankfurt/M. 71

Commodore AG
Aeschenvorstadt 57
CH-4010 Basel

Commodore GmbH
Fleschgasse 2
A-1130 Wien

Änderungen vorbehalten

Stand März 1982

Artikel-Nr. 580020/11-83