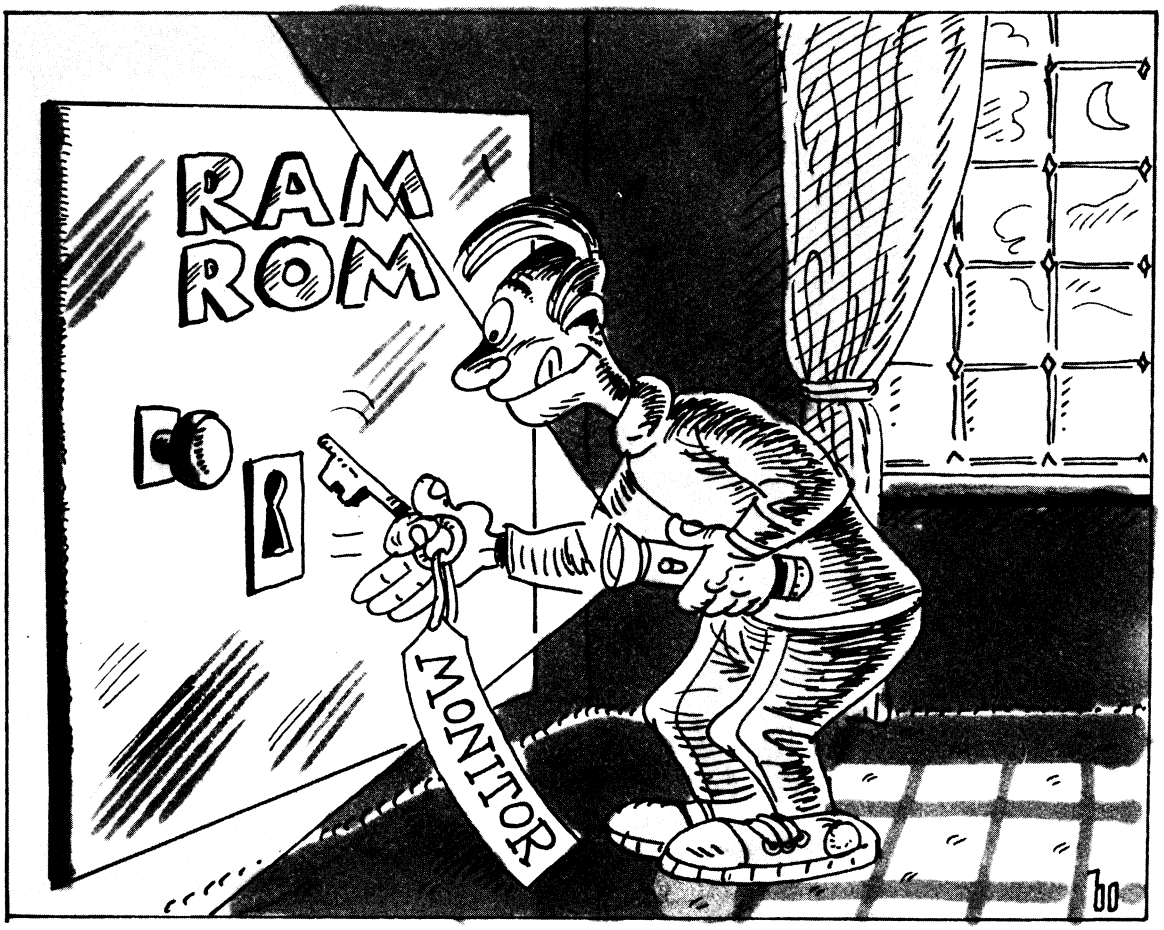
Durchblick mit dem Monitor



Vielen C16/116-Anwendern ist der eingebaute Maschinensprache-Monitor TEDMON immer noch ein Buch mit sieben Siegeln. Hier soll Ihnen gezeigt werden, was man mit ihm als Basic-Programmierer alles anfangen kann.

Leider wird der TEDMON im C16-Handbuch nur für Maschinensprache-Freaks erklärt. Basic-Programmierer werden praktisch auf dem Trockenen sitzen gelassen. Das aber eigentlich zu Unrecht, denn auch der Basic-Programmierer kann mit diesem Werkzeug eine ganze Menge anfangen – auch ohne Kenntnisse in Maschinensprache.

Was ist überhaupt ein Monitor? Diese Frage hat zwei Antworten. Die erste kennen Sie bestimmt: Ein Sichtgerät, an das Sie Ihren Computer anschließen können. Die zweite Antwort ist: Keine Hardware, sondern ein Programm, mit dem man einen Einblick in den Speicher des Computers bekommt.

Im C16, C116 und Plus/4 ist ein solches Monitorprogramm schon integriert: der TEDMON. Bevor wir mit diesem arbeiten können, müssen wir uns noch ein wenig Grundlagenwissen aneignen.

Die grundlegende Fähigkeit eines Computers ist es, Daten erfassen zu können. Man spricht beim Menschen davon, daß er sich etwas merkt. Beim Computer sagt man, er speichert etwas. Daher kommt der Begriff Speicher.

Der Speicher benötigt eine bestimmte Struktur, um zweckmäßig arbeiten zu können. Wir schlüsseln diese Unterteilung in Speichereinheiten »von unten« auf, beginnen also mit der kleinsten Speichermenge. Die einfachste Art von Information ist die Unterscheidung zwischen zwei Möglichkeiten (ja/nein, gut/schlecht, und so weiter).

Beim Computer, der sich Speicherwerte (vereinfacht gesagt) durch Verändern der elektrischen Spannung merkt, wird zwischen »Strom fließt« und »Strom fließt nicht« unterschieden. Dabei steht »0« für »kein Strom« und »1« für »Strom«.

Diese primitivste Form der Information nennt man sowohl in der Informatik (»Computerlehre«) als auch in der modernen Informationstheorie »Bit«. »BIT« kommt vom englischen »BInary digiT« (Binärziffer), da in einem Zahlensystem der Basis 2 nur 0 und 1 als Ziffern vorhanden sind (zum Vergleich: Wir arbeiten im alltäglichen Gebrauch mit der Basis 10, dem Dezimalsystem).

Natürlich ist ein Bit zu wenig, um vernünftig arbeiten zu können. Deshalb kombiniert man mehrere Bits, das heißt man nimmt mehrere Bits zu einer höheren Informationseinheit zusammen und betrachtet jedes einzelne Bit als Stelle einer Binärzahl. Mathematisch gesehen haben wir dann »2 hoch Anzahl der Bit«-Kombinationen.

Computerhersteller und -entwickler kamen zu der Lösung, 8 Bit als ein »Byte« zu bezeichnen. Wir können also 2^8, das sind 256 verschiedene Werte darstellen. Hier ein paar Beispiele:

binär %01010101 = dezimal 85
binär %11111111 = dezimal 255 (höchste Zahl, die mit 8 Bit dargestellt werden kann)
binär %00000000 = dezimal 0 (niedrigste mit 8 Bit darstellbare Zahl)
binär %10001000 = dezimal 136

Ein Byte kann alle Zahlen von 0 bis 255, also 256 verschiedene Werte, annehmen. Eine »Speicherzelle« beim Computer hat genau 1 Byte Kapazität. Bei den Beispielen der Binärzahlen ist Ihnen vielleicht das Prozentzeichen »%« vor jedem Binärwert aufgefallen. Dieses Zeichen setzt man in der Regel vor eine Binärzahl, um damit das Zahlensystem der Basis 2 zu kennzeichnen

Das Zusammenfassen von Bits in einem Byte erleichtert uns die Einteilung des Speichers erheblich; man darf aber nicht außer acht lassen, daß für den Computer nach wie vor der gesamte Inhalt des Speichers nur eine Ansammlung sehr vieler Bits ist.

Auch Buchstaben und andere Zeichen werden vom Computer als Byte behandelt. Dafür gibt es eine Codierung, nach der jedem Byte-Wert ein Zeichen zugewiesen ist, zum Beispiel steht die Zahl 65 (dezimal) für den Buchstaben A, 66 für B, 67 für C, etc.. Diese Codierung trägt die Bezeichnung »ASCII« (»American Standard Code for Information Interchange« oder deutsch »Amerikanische Standard-Codierung für den Informationsaustausch«). Man sagt also zum Beispiel: »Der ASCII-Code von A ist 65«. Soviel zunächst zum ASCII-Code; dazu können Sie sich auch im Handbuch zu Ihrem Computer informieren.

Natürlich ist ein einziges Byte zu wenig. Von einem Computer erwartet man, daß er viele Zeichen aufnehmen kann. Unser Computer hat zum Beispiel über 16000 Byte Speicherkapazität, wenn es sich um einen C16 oder C116 handelt. Der Plus/4 verfügt sogar über ganze 65536 Byte Speicher.

Damit der Computer »weiß«, welche Speicherstelle angewählt (zum Lesen oder zum Schreiben) werden soll, erhält jede Speicherzelle eine ihr zugewiesene Nummer, die sogenannte »Speicheradresse« oder einfach nur »Adresse«. Mit der Nummer 1024 wird also beispielsweise die 1025. Speicherstelle im Computer angesprochen. 1025 deshalb, weil die Zählung im Computer immer mit Null beginnt. Die erste Speicherstelle trägt also die Adresse 0.

Eine Einheit von 2^{10} = 1024 Byte nennt man »Kilo-Byte« oder kurz »KByte«. Die Mengeneinheit »Kilo« steht also nicht für 1000, sondern für 1024 Byte. Das ist übrigens eine häufige Fehlerquelle. Wenn man von einem Computer mit 64 KByte Speicher spricht, dann handelt es sich nicht um 64000 Byte, sondern um 64 * 1024 = 65536 Byte Speicherplatz. Da der Computer intern binär arbeitet, also keine 1000 als gerade Potenz (im Dezimalsystem 10^3) kennt, ist diese Einteilung zweckmäßig.

Da unser C16 in der Grundausführung 16 KByte sogenanntes »RAM« zur Verfügung hat, sind das also 16 * 1024 = 16384 Byte.

Was ist »RAM«?

Wir sprachen eben von »RAM«. Das bedarf einer kurzen Erklärung. RAM (vom englischen »Random Access Memory« oder auf deutsch etwa »Speicher mit wahlfreiem Zugriff«) ist der Speicher, dessen Inhalt man im Computer verändern kann. Ein Basic-Programm oder Ihre Daten werden beispielsweise im RAM abgelegt beziehungsweise gespeichert.

Die andere Sorte von Speicher ist das ROM (»Read Only Memory«), der Nur-Lese-Speicher. Im ROM stehen bestimmte Daten fest eingespeichert (die elementaren Steuerprogramme des Computers, die so etwas wie Bildschirmausgabe oder Basic-Programmierung erst ermöglichen) und können zwar gelesen, aber nicht geändert werden. Der Vorteil des ROM gegenüber des RAM ist, daß die dort enthaltenen Werte auch nach dem Ausschalten der Stromzufuhr erhalten bleiben. Der Vorteil des RAM ist, daß man freien Zugriff (sowohl Lesen als auch Schreiben von Speicherinhalten) auf den Speicher hat.

Im Computer werden dabei in der Regel beide Möglichkeiten realisiert. Das RAM steht dem Anwender für seine Arbeit zur Verfügung (bis auf wenige Ausnahmen), und das ROM enthält das Steuerprogramm, das den Computer nach dem Einschalten in einen kontrollierten Zustand bringt. Dieses Steuerprogramm nennt man auch Betriebssystem, wobei dieses Betriebssystem für die elementaren Vorgänge, wie Schreiben auf den Bildschirm, Eingabe von der Tastatur, Laden und Speichern von Programmen, zuständig ist.

Das Hexadezimalsystem

Wir wissen nun, daß der Computer intern binär rechnet, auch wenn er in Basic uns gegenüber so tut, als ob er dezimal arbeitet. Für diesen Komfort sind extra Rechenprogramme verantwortlich, die im ROM des Computers gespeichert sind.

Der Monitor führt uns aber näher an die Maschinenebene heran, so daß wir uns dem Computer ein wenig anpassen müssen. Es wäre jedoch zuviel verlangt, wenn wir jetzt auf einmal in Nullen und Einsen denken sollten (vor wenigen Jahren war das jedoch noch so!). Andererseits müssen wir aber auf den Komfort unter Basic verzichten.

Der Kompromiß ist ein Zahlensystem, das leicht vom Computer ins Binärsystem umgewandelt, aber auch von uns leicht verwendet werden kann. So »erfand« man das Hexadezimalsystem, ein Zahlensystem mit der Basis 16. Die Vereinfachung ist dabei, daß je 4 Bit einer hexadezimalen Ziffer entsprechen. 1 Byte (= 8 Bit) kann also aus maximal 2 Hex-Ziffern bestehen ($2^8 = 16^2$).

Es stehen die vom Dezimalsystem her bekannten Ziffern 0 bis 9 weiterhin zur Verfügung; außerdem brauchen wir noch Ziffern, die den dezimalen Werten 10 bis 15 entsprechen. Hierfür stehen die Buchstaben A bis F. Es ergibt sich für Hexadezimal-Ziffern folgende Umrechnungstabelle:

Hexadezimal	Dezimal	Hexadezimal	Dezimal
0	0	8	8
1	1	9	9
2	2	A	10
3	3	B	11
4	4	C	12
5	5	D	13
6	6	E	14
7	7	F	15

Treffen wir an dieser Stelle eine Vereinbarung: Um Dezimalzahlen von Hexadezimal-Zahlen (auch: Hex-Zahlen) zu unterscheiden, stellen wir den Hex-Zahlen immer ein Dollarzeichen »$« voran. $44 ist also hexadezimal, 44 (ohne Dollar davor) dezimal. Dieses System kennen Sie aber schon von den Binärzahlen her, als wir ein Prozentzeichen »%« zur Unterscheidung verwendeten.

Selbstredend können Hex-Zahlen auch aus mehreren Stellen bestehen.

Im Dezimalsystem ist:

$345 = 5 * 10^0 + 4 * 10^1 + 3 * 10^2$

Zur Probe:

$5 * 10^0 + 4 * 10^1 + 3 * 10^2$
$= 5 * 1 + 4 * 10 + 3 * 100$
$= 5 + 40 + 300$
$= 345$

Ähnlich gehen wir im Hexadezimalsystem vor, allerdings ist die Basis jetzt 16 und wir müssen die Hex-Ziffern umrechnen:

$\$3A4 = \$4 * \$10^0 + \$A * \$10^1 + \$3 * \$10^2$
$= 4 * 16^0 + 10 * 16^1 + 3 * 16^2$
$= 4 * 1 + 10 * 16 + 3 * 256$
$= 4 + 160 + 768$
$= 932$

Beim Arbeiten mit dem Monitor wird alles über das hexadezimale Zahlensystem abgewickelt: Der Monitor gibt Werte (Byte, Adressen) hexadezimal aus, wir geben Werte hexadezimal ein.

Sollte einmal eine Umrechnung anfallen, so greifen wir auf Basic zurück. Den hexadezimalen Wert einer Dezimalzahl erhalten wir über

```
PRINT HEX$(Dezimalzahl)
```

zum Beispiel

```
PRINT HEX$(952)
```

Die gegensätzliche Umrechnung erfolgt über

```
PRINT DEC("Hex-Zahl")
```

zum Beispiel

```
PRINT DEC("5FA3")
```

Dabei sind Werte von 0-65535 beziehungsweise $0000-$FFFF zugelassen, andere Werte benötigen wir auch nicht.

Der Monitor kann auf drei Arten gestartet werden, von denen nur zwei für uns interessant sind:

1) Über den Maschinensprachebefehl »BRK«. Diese Variante vernachlässigen wir, denn uns interessiert hier nur Basic.

2) RESET und gleichzeitig RUN/STOP-Taste gedrückt halten. Dies bewirkt, daß beim RESET nicht ins Basic, sondern in den Monitor gesprungen wird.

3) Eingabe des Basic-Befehls <MONITOR>.

Mit welcher der drei genannten Möglichkeiten der Monitor auch gestartet wird, er meldet sich mit einer Einschaltmeldung:

```
MONITOR
  PC   SR AC XR YR SP
; 0000 00 00 00 00 F8
```

»MONITOR« signalisiert, daß wir uns nun im Monitor befinden. Jetzt haben alle Basic-Kommandos ihre Gültigkeit verloren, bis wir den Monitor verlassen. Dafür gelten nun die Monitor-Anweisungen.

Start des Monitors

Die weiteren Zeilen der Einschaltmeldung können wir vernachlässigen, denn die sind nur für Maschinensprache-Anwendungen interessant. In diesem Kurs werden wir aber nichts besprechen, was nur für Maschinensprache Bedeutung hat; wer Maschinensprache kann, findet im 64'er-Sonderheft 3/1986 ab Seite 14 und auch in diesem Sonderheft eine ganze Menge Informationen darüber.

Der Eingabemodus des Monitors ist nicht anders als in Basic, Cursortasten und so weiter funktionieren weiterhin.

Nur die Basic-Anweisungen und Zeilennummern gibt es jetzt nicht mehr.

Was versteht er aber dann für eine »Sprache«?

Dazu wollen wir gleich unseren ersten Befehl lernen. Er dient zum Verlassen des Monitors (ins Basic) und heißt ganz einfach »X«. Sie wissen schon, wie man dieses Kommando ausführt: <X> eingeben und <RETURN> drücken. Das vertraute »READY.« erscheint und wir sehen daran, daß wir uns im Basic befinden. Nun können wir wieder Basic-Befehle ausführen lassen. Ein Neustart des Monitors erfolgt über <MONITOR>.

Dem Befehl »X« müssen keine weiteren Angaben (sogenannte Parameter) folgen, der Monitor weiß auch so, was er zu tun hat.

In Bild 1 finden Sie eine Kurzübersicht zum X-Befehl. Wir werden Ihnen zu jedem neuen Befehl eine solche Kurzübersicht liefern, damit Sie bei Bedarf schnell nachschlagen können.

Übrigens bestehen alle Monitor-Kommandos – wie der X-Befehl – nur aus einem einzigen Zeichen. Dieses Zeichen ist meist aus einem englischen Wort, das in der Kurzübersicht aufgeführt ist, abgeleitet.

Der nächste Befehl, den wir kennenlernen, ist nicht so einfach wie »X« anzuwenden. Dafür eröffnet uns dieses Kommando aber auch wesentlich mehr Möglichkeiten. Es ist der M-Befehl, der »Grundbefehl« eines Monitors. Wenn wir uns mit diesem vertraut gemacht haben, sind die restlichen Befehle schnell erlernt. Lassen Sie sich beim Verstehen des M-Befehls also Zeit, es lohnt sich!

Zunächst etwas Grundsätzliches: Mit <RUN/STOP> können fast alle Monitor-Befehle abgebrochen werden. Bei »X« ist dies deswegen nicht möglich, weil er augenblicklich ausgeführt wird.

Der Befehl »MEMORY DUMP«

Die Hauptaufgabe eines Monitorprogramms ist es, Speicherbereiche anzuzeigen. Das geschieht natürlich hexadezimal.

Wir geben dem Monitor dazu die erste und die letzte anzuzeigende Adresse an. Geben Sie bitte folgenden Befehl ein (gefolgt von <RETURN>):

```
M 8180 81B5
```

Sie erhalten die in Bild 2 zu sehende Antwort des Monitors. Sollten Sie andere Werte erhalten, geben Sie zunächst

```
07F8 00
```

und dann wieder

```
M 8180 81B5
```

ein.

Die Anzeige bezeichnet man als »Memory Dump«, auf deutsch: »Speicherausdruck«. Ohne Erklärung kann man mit diesem hexadezimalen Zahlen- und Zeichen-Gewirr beim besten Willen nichts anfangen. Betrachten wir deshalb exemplarisch die erste Zeile (>8180 0A A9 ...).

Das Größer-Zeichen (>) am Anfang der Zeile ist erst an späterer Stelle von Bedeutung.

Die 4stellige Hex-Zahl danach, nämlich 8180, ist die sogenannte »Basisadresse« der Dump-Zeile. In dieser Zeile werden also die Inhalte der Speicherzellen ab 8180 angezeigt.

Diese Inhalte der Adressen stehen nach der Adresse in Form von 8 zweistelligen Hex-Zahlen. Nun benötigen wir besagte Basisadresse, um sagen zu können, welche Zahl der Inhalt welcher Speicherzelle ist. Die 8 Byte sind ganz einfach die Speicherinhalte ab der Basisadresse:

$0A ist der Inhalt von $8180, $A9 von $8181, $50 von $8182, und so weiter.

Nach dem Doppelpunkt stehen in der Reihenfolge der Bytes die Zeichen, zu denen die Bytes die ASCII-Codes, beziehungsweise die Commodore-Codes darstellen. Commodore-Codes deshalb, weil sich Commodore mit seinen vielen Sonderzeichen nur sehr begrenzt an die ASCII-Codierung hält. Wie wir wissen, hat jedes Zeichen einen Code, der aus einer Zahl von $00 bis $FF besteht. ASCII kennt aber nur Zeichen-Codes von $00 bis $7F. Bei unserem Computer benötigen wir deshalb einen eigenen Code, was uns aber im weiteren nicht stören soll.

Da Zeichen im Speicher als Zahlen abgelegt werden, zeigt unser Monitor zu jedem Byte den Code an, damit wir sofort sehen können, wo Texte stehen. In unserem Beispiel haben wir die Codes einiger Basic-Befehle erwischt.

Sie werden mit "M 8000 FFFF" auch noch andere Texte, die der Computer ausgibt, finden.

Bei der Code-Darstellung muß der Monitor noch ein kleines Problem lösen. Die Zeichen-Codes gibt es nämlich nicht nur für Buchstaben, Ziffern, Grafikzeichen und so weiter, sondern auch für sogenannte Steuerzeichen. Steuerzeichen sind im Gegensatz zu den »druckbaren Zeichen« diejenigen Codes, die kein Zeichen am Bildschirm ergeben, sondern eine Steuerfunktion hervorrufen. Mit <PRINT CHR$(65)> erhalten wir das druckbare Zeichen »A«, mit <PRINT

CHR$(147)> jedoch zum Beispiel das Steuerzeichen für »Bildschirm löschen«.

Da Steuerzeichen also nicht darstellbar sind, werden Sie vom Monitor in der Zeichen-Darstellung als Punkt ».« ausgegeben.

Es ist allerdings zu beachten, daß es den Punkt auch als richtiges Zeichen gibt. Dieser »echte« Punkt hat den ASCII-Code $2E oder dezimal 46 (probieren Sie in Basic PRINT CHR$(46)) und ist beim Memory Dump nicht von den Steuerzeichen zu unterscheiden. Im Zweifelsfall muß man also auf die Hex-Werte sehen, um erkennen zu können, ob es sich um einen wirklichen Punkt (Wert: $2E) oder ein Steuerzeichen (anderer Wert als $2E) handelt.

Am Anfang ist es schwer einzusehen, weshalb der M-Befehl so wichtig ist. Bald werden wir aber praktische Anwendungen besprechen, damit Sie einen ersten Eindruck von der Mächtigkeit dieses Monitor-Kommandos bekommen.

Zunächst müssen wir jedoch noch die Syntax des M-Befehls betrachten:

In unserem Beispiel (»M 8180 81B5«) haben wir die volle Syntax verwendet:

M Anfangsadresse Endadresse

Wenn wir die Endadresse weglassen, ergibt sich die Syntax »M Anfangsadresse«. Dann wird ab der angegebenen Anfangsadresse ein Memory Dump von 12 Zeilen Länge ausgegeben. Da in jeder Zeile 8 Adressen gedruckt werden, werden beim Weglassen der Endadresse also 12*8 = 96 Byte ausgegeben. Die Adressen 0 bis 96 des C16-Speichers lassen Sie bitte über <M 0000> ausgeben, damit Sie einmal den Effekt gesehen haben.

Es gibt sogar noch eine dritte Syntax: <M> (ohne weitere Angaben danach).

Dann wird ab der letzten angezeigten Adresse 12 Zeilen lang gedruckt. Wenn Sie eben wie gefordert <M 0000> eingegeben haben, werden jetzt auf ein nachfolgendes <M> hin die Speicherzellen ab $0060 ausgegeben.

Der Befehl »M« ohne Parameter ist vor allem dann nützlich, wenn man das Dump zuvor durch <RUN/STOP> abgebrochen hatte.

In Bild 3 finden Sie die Kurzübersicht zum M-Befehl. Wir wollen nun zu ersten Anwendungen kommen.

Anwendung 1: Basic-Anfang und Basic-Ende

Wenn man ein Basic-Programm im Speicher hat, möchte man gerne wissen, wo es beginnt und wo es aufhört. Hat man diese Grenzen erst einmal ermittelt, kann man das Basic-Programm über »M« ausgeben lassen.

In $002B/$002C befindet sich die Anfangsadresse des Programms. Diese muß auf zwei Adressen ($002B UND $002C) aufgeteilt werden, da eine Adresse hexadezimal vierstellig ist, eine Speicherstelle aber nur zweistellige Hex-Zahlen aufnehmen kann. Der Computer bedient sich hier eines einfachen, aber wirkungsvollen Tricks: Die Speicherzelle mit der niedrigeren Adresse ($002B) erhält die beiden niederwertigen Stellen der vierstelligen Zahl, die andere Adresse ($002C) die beiden höherwertigen Stellen. Die niederwertigen Stellen nennt man »Low-Byte«, die höherwertigen »High-Byte«.

Ein Beispiel: Die Zahl $1AB5 soll in Low- und High-Byte aufgeteilt werden. Das Low-Byte besteht aus den niederwertigen (weiter rechts stehenden) Stellen und ist also $B5, das High-Byte entsprechend $1A.

In $002B liegt das Low-Byte, in $002C das High-Byte der Anfangsadresse des Basic-Programms im Speicher. Wir stellen jetzt diese Anfangsadresse fest.

Geben Sie dazu <M 002B 002C> ein. In 99 Prozent aller Fälle erhalten wir:

>002B 01 10 ...

Das Low-Byte ist somit $01, das High-Byte $10. Wie wir wissen, besteht das Low-Byte aus den beiden rechts stehenden Stellen, das High-Byte aus den beiden links stehenden. Richtig gestellt ist die Adresse also $1001. Wir hätten auch so vorgehen können, daß wir die Reihenfolge im Speicher (»01 10«) einfach umstellen und »1001« erhalten. Mit dem Basic-Befehl PRINT DEC ("1001") können wir unser »Forschungsergebnis« ins dezimale Format umwandeln lassen, wenn wir wollen.

Die Endadresse stellen wir auf gleiche Weise fest. Wir geben <M 002D 002E> ein und erhalten jetzt Werte, die von der Länge des aktuellen Basic-Programms abhängen. Jedenfalls nehmen wir die erste zweistellige Zahl und stellen die Ziffern der zweiten zweistelligen Zahl davor; das Ergebnis ist die Endadresse. War kein Programm im Speicher (zum Beispiel direkt nach dem Einschalten oder nach »NEW« der Fall), müßten Sie $1003 erhalten.

Anwendung 2: Basic-Programm ansehen

Laden Sie nun ein Basic-Programm ein und ermitteln Sie Anfangs- und Endadresse wie in Anwendung 1. Mit <M Anfangsadresse Endadresse> ist es dann möglich, das Basic-Programm im Speicher anzusehen. Warum dort wel-

```
Befehl:        X ("eXit")
Syntax:        X
Wirkung:       Rücksprung vom Monitor ins Basic
```

Bild 1. Kurzübersicht zum Befehl »X«

```
>8180 0A A9 50 D0 06 A9 6C D0  :
>8188 02 A9 5A 4C 94 04 45 4E  :
>8190 C4 46 4F D2 4E 45 58 D4  :
>8198 44 41 54 C1 49 4E 50 55  :
>81A0 54 A3 49 4E 50 55 D4 44  :
>81A8 49 CD 52 45 41 C4 4C 45  :
>81B0 D4 47 4F 54 CF 52 55 CE  :
```

Bild 2. Beispiel für die Anwendung des M-Befehls

```
Befehl:        M ("Memory dump")
Syntax:        M Anfangsadr. Endadr.
          oder M Anfangsadr.
          oder M
Wirkung:       Anzeige der Adresse und ihrer
               Speicherinhalte
```

Bild 3. Kurzübersicht zum Befehl »M«

```
Befehl:        >
Syntax:        > Adresse Byte1 Byte2 ByteX
Wirkung:       Die Bytes werden ab der
               Adresse abgelegt
```

Bild 4. Kurzübersicht zum >-Befehl

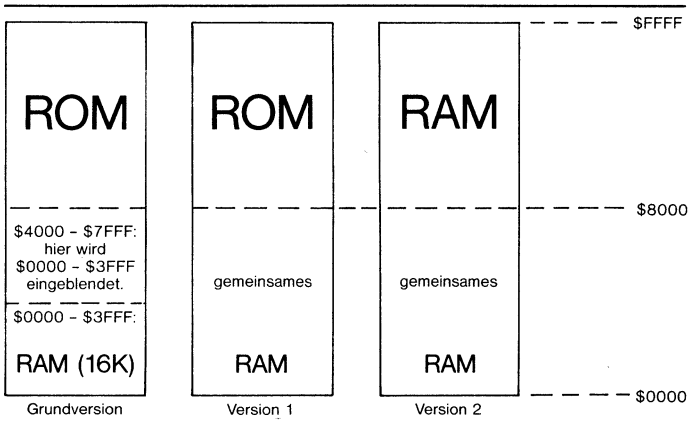


Bild 5. Bank-Switching. Im Bereich $0000 bis $7FFF ist immer RAM eingeblendet

che Werte stehen, besprechen wir am Ende des Kurses. Durch dieses Ansehen können Sie aber schon einen ersten Vorgeschmack bekommen.

Anwendung 3: Bildschirmspeicher betrachten

Diese Anwendung ist zwar unproduktiv, aber eine ganz gute Demonstration des M-Befehls.

Mit <M 0C00 0FE0> können wir den Bildschirmspeicher (das ist der Speicherbereich, in dem der Bildschirminhalt untergebracht ist) ausgeben. Wie Sie daraus ersehen können, wird auch der Bildschirm vom Computer wie ein normales RAM behandelt. Lediglich der Videoprozessor holt sich daraus seine Daten für das Aufbereiten des Bildes.

Sehen Sie sich das am besten einmal an.

Speicherzellen mittels Memory Dump ändern

Es ist interessant, die Speicherzellen mittels <M> anzusehen, und es vermittelt das Gefühl, hinter die Kulissen zu blicken.

Man verspürt aber auch das Bedürfnis, in das Geschehen einzugreifen, indem man die Inhalte der Speicherzellen ändert. Auch dafür kann man das Memory Dump einsetzen.

Wollen wir die angezeigten Werte ändern, müssen wir nur mit dem Cursor in die betreffende Dump-Zeile fahren, die hexadezimalen Werte mit unseren neuen Werten, die auch zweistellig hexadezimal sein müssen, überschreiben und <RETURN> drücken. Damit Sie es gleich ausprobieren können, geben Sie <M 1000> ein. Sie erkennen dort den Anfang des im Speicher befindlichen Basic-Programms (gegebenenfalls Monitor verlassen, Programm über Basic laden und dann den Monitor neu starten).

Gehen Sie mit dem Cursor in die erste Dump-Zeile und dort auf das erste Byte (nicht auf die Basisadresse, sondern auf die zweistellige Zahl danach!). Das erste Byte ist sicher »00«. Überschreiben Sie diesen Wert mit »5A« und drücken Sie dann <RETURN>.

Wie Sie sehen, wird die Änderung übernommen, denn in der ASCII-Darstellung verschwindet der Punkt, der das erste Byte repräsentierte, und es erscheint ein »Z« (das liegt am Wert $5A, dem ASCII-Code für »Z«). Verlassen Sie nun über <X> den Monitor und versuchen Sie, das Programm über <RUN> zu starten. Der Computer meldet einen »?SYNTAX ERROR«. Der Grund dafür ist die Änderung der Adresse $1000. Wenn wir sie wieder rückgängig machen (<M 1000>; erstes Byte mit »00« überschreiben; <RETURN> drücken), können wir das Programm auch wieder starten: <RUN> ergibt keinen »?SYNTAX ERROR« mehr.

Dieses Beispiel sollte nur eine kleine Demonstration sein, um das Prinzip zu verdeutlichen. Sie können in einer Zeile auch mehrere Bytes (im Extremfall alle) ändern; in jedem Fall dürfen Sie aber nur die Hex-Darstellung ändern (die ASCII-Darstellung können Sie zwar überschreiben, Ihre Änderung wird aber nicht in den Speicher übernommen).

Beim Ändern der Speicherinhalte wird die ASCII-Darstellung vom Computer nicht beachtet. Deshalb steht nach den Hex-Bytes der Doppelpunkt, der dem Monitor signalisiert, daß alles, was dem Doppelpunkt folgt, ignoriert werden soll.

Es ist also recht einfach, Speicherinhalte zu ändern. Man muß nur den entsprechenden Bereich ausgeben und dann die zu ändernden Bytes mit den gewünschten Werten überschreiben.

Das Größer-Zeichen (>) am Anfang einer Dump-Zeile ist nichts weiter als ein eigener Monitor-Befehl. Die Syntax lautet:

```
>Basisadresse Bytes
```

Die Basisadresse ist die Adresse, ab der die Speicherinhalte geändert werden sollen. Ab dieser Basisadresse werden dann die der Adresse folgenden Bytes abgelegt. Wir können nur ein Byte, aber auch mehrere Byte schreiben lassen.

Der Befehl >0C00 36 34 27 45 52 schreibt $36 in Adresse $0C00, $34 in Adresse $0C01 und so weiter.

Da diese Adressen der Anfang des Bildschirmspeichers sind, sehen Sie links oben am Bildschirm den Text »64'ER«.

Eine andere Anwendung wäre >FF19 00 (färbt den Rahmen schwarz).

Bild 4 ist die Kurzübersicht zum Befehl (>).

Den >-Befehl können wir auch zum sogenannten Bank-Switching einsetzen. Dazu müssen wir erst einmal klären, was Bank-Switching eigentlich ist.

Der Prozessor unseres C16/116 und des Plus/4 kann nur 2^{16} = 65536 Adressen anwählen. In der Grundversion (von C16/116) reicht das:

RAM	16 KByte
ROM	32 KByte
insgesamt	48 KByte

in Bytes (48*1024) = 49152

Der Speicher ist dann wie folgt aufgebaut: Von $0000 bis $4000 stehen die 16 KByte RAM, im Bereich $8000 bis $FFFF die 32 KByte ROM. Der Bereich $4000 bis $8000 ist genaugenommen ungenutzt, denn dort wird nur der RAM-Bereich $0000 bis $4000 eingeblendet. $4005 ist also nichts anderes als $0005. Diese Angaben gelten jedoch nur für die Grundversion von C16 und C116.

In diesem Fall kann man sagen, daß der Computer mit der Verwaltung des Speichers unterfordert ist, denn er könnte noch weitere 16 KByte adressieren.

Bei einer 64-KByte-Speichererweiterung sind jedoch 64 KByte RAM und 32 KByte ROM, also 96 KByte Speicher, zu adressieren (zum Beispiel im Plus/4).

Hier hat der Computer die Lösung des Bank-Switching parat. Es wird dabei zwischen zwei Adreßkonfigurationen umgeschaltet, in denen jeweils 64 KByte »sichtbar« sind:

Version 1 (ROM-Lösung)

$0000-$7FFF	RAM
$8000-$FFFF	ROM

In dieser Version wird nur ein Teil vom RAM (nur 32 KByte statt der gesamten 64 KByte) verfügbar gemacht; dafür aber ist das komplette ROM ab $8000 eingeblendet.

Version 2 (RAM-Lösung)

$0000-$FFFF	RAM

Das ROM wird ausgeblendet, dafür sind die gesamten 64 KByte RAM verfügbar.

Wir können jeweils nur eine Konfiguration einschalten. Verantwortlich hierfür ist die Adresse $07F8, die in beiden Versionen als RAM verfügbar ist.

Steht in $07F8 der Wert $00, haben wir Version 1 eingeschaltet, steht der Wert $80, so ist Version 2 gültig. So können wir mittels Umschalten auf jede Adresse im Speicher durch den Monitor zugreifen.

Version 1 ist übrigens der Einschaltzustand; wollen wir nur mit Version 1 arbeiten, müssen wir nie umschalten. Dies ist zum Beispiel dann der Fall, wenn Sie die Grundversion des C16/116 haben, in der ja ein Bank-Switching nicht nötig ist.

Die Monitor-Befehle zum Umschalten sind:

```
>07F8 00: schaltet Version 1 ein
>07F8 80: schaltet Version 2 ein
```

In Bild 5 finden Sie noch eine grafische Darstellung des Bank-Switching.

Gezielte Suche nach Bytes

Wenn man den Speicher bearbeitet, möchte man vielleicht manchmal bestimmte Werte suchen. Dafür gibt es einen Suchbefehl.

Mit <H 8000 FFFF 49 4E D0> suchen wir im Bereich von $8000 bis $FFFF (natürlich in der aktuellen Speicherkonfiguration) nach der Bytefolge $49 $4E $D0.

Wir erhalten vom Monitor die Antwort:

```
8159 8165
```

Das bedeutet, daß die von uns gesuchte Bytefolge ab $8159 und ab $8165 im Speicher steht.

Wenn wir nach Zeichen suchen wollen, geht dies auch. Dazu müssen wir nach Anfangs- und Endadresse des zu durchsuchenden Bereichs ein Apostroph <'> eingeben und danach die Suchzeichen. Ein Beispiel:

```
H 8000 FFFF 'COMMODORE
```

sucht nach dem Text »COMMODORE« im Speicher von $8000 bis $FFFF. Wir erhalten als Antwort:

```
80CF E3A7
```

Falls Sie bei unseren beiden Beispielen zum H-Befehl nicht die genannten Adressen erhalten haben, haben Sie nicht die Speicherkonfiguration von Version 1 eingestellt, was Sie über <>07F8 00> leicht nachholen können.

Die Kurzübersicht zum H-Befehl ist in Bild 6 dargestellt.

An dieser Stelle soll Ihnen noch ein wichtiger Hinweis zum H-Befehl gegeben werden, der dringend erforderlich ist:

Wenn Sie den zu durchsuchenden Bereich so wählen, daß auch der Bereich $0200 bis $02FF geprüft wird, sind alle Werte von $0200 bis $025D, die der H-Befehl liefert, zu ignorieren. Der Grund ist, daß Werte aus diesem Bereich keine Aussagekraft haben.

Bei der C 16-Grundversion sind auch Werte von $4200 bis $425D zu vernachlässigen, da es sich nur um Kopien von $200 bis $25D handelt.

Die ASCII-Codes

Im Handbuch finden Sie zwar eine Tabelle der ASCII-Codes, aber diese führt nur den dezimalen Commodore-ASCII-Code auf.

Mit dem Programm in Listing 1 können Sie sich eine hexadezimale Tabelle ausgeben lassen. Diese Tabelle stellt bei den ASCII-Codes $00 bis $1F und $80 bis $9F kein Zeichen dar, da es sich hierbei um Steuerzeichen handelt.

Bei $20, $A0 und $E0 steht auch kein Zeichen, denn diese Codes stehen für das Leerzeichen (»Space«).

Wenn Sie bei der Arbeit mit dem Monitor den Commodore-ASCII-Code für ein bestimmtes Zeichen wissen wollen, nehmen Sie die Tabelle zur Hand und suchen das Zeichen. Dann finden Sie die erste hexadezimale Stelle in der waagrechten Spaltennumerierung von 0 bis F, die zweite Hex-Stelle in der senkrechten Numerierung der Zeilen (auch von 0 bis F). Nehmen wir als Beispiel das Ausrufezeichen »!«: die erste Stelle ist 2, die zweite ist 1. Der hexadezimale ASCII-Code ist also $21.

So läßt man die Umrechnung über Basic ausführen:

```
PRINT RIGHT$(HEX$(ASC("A")),2)
```

ergibt den hexadezimalen ASCII-Code des Buchstabens »A« als zweistellige Zahl; andere ASCII-Codes erhalten Sie, wenn Sie statt »A« andere Zeichen einsetzen.

Bei manchen Zeichen geht die Umrechnung ganz einfach. Der ASCII-Code einer Ziffer hat als linke Hex-Ziffer die »3« und als rechte Stelle die Ziffer selbst; der ASCII-Code von 5 ist also $35.

Ähnlich leicht errechnet sich der ASCII-Code der Zeichen, die über <SHIFT> und eine Zahlentaste erreicht werden (zum Beispiel das Ausrufezeichen <SHIFT 1>, Und-Symbol <SHIFT 6>); man nimmt die »2« als linke Hex-Ziffer und als rechte Stelle verwendet man die Zahlentaste, die man außer <SHIFT> drücken muß, um das Zeichen zu erhalten.

Das raffinierteste Umrechnungsverfahren aber bedient sich nur des Monitors.

Wir rechnen damit den Text »BEISPIEL« in den ASCII-Code um. Dazu geben wir zunächst

```
H 0 0 'BEISPIEL
```

ein. Dieser Suchbefehl bringt keine Adresse auf den Bildschirm, das soll er aber auch gar nicht. Nun steht im Suchpuffer des Monitors dieser Text im ASCII-Code. Folgender Befehl

```
Befehl:      H ("Hunt")
Syntax:      H Anfangsadr. Endadr. Byte1 ... ByteX
       oder  H Anfangsadr. Endadr. 'Text
Wirkung:     Suchen der Bytes oder des Textes im
             durch Anfangsadr. und Endadr.
             eingegrenzten Bereich
```

Bild 6. Kurzübersicht zum H-Befehl

ASCII-Code	Wirkung
$05	Cursorfarbe weiß
$08	SHIFT C = verbieten
$09	SHIFT C = wieder zulassen
$0D	RETURN (Cursor an Anfang der nächsten Zeile)
$0E	Klein/Großschrift ein
$11	Cursor eine Zeile nach unten
$12	Reversschrift ein
$13	Cursor in linke obere Ecke
$14	letztes Zeichen löschen
$1B	ESC
$1C	Cursorfarbe rot
$1D	Cursor eine Spalte nach rechts
$1E	Cursorfarbe grün
$1F	Cursorfarbe blau
$82	FLASH ON (Blinken ein)
$83	FLASH OFF (Blinken aus)
$8D	SHIFT RETURN (wie $0D)
$8E	Groß/Grafikschrift ein
$90	Cursorfarbe schwarz
$91	Cursor eine Zeile nach oben
$92	Reversschrift aus
$93	Bildschirm löschen
$94	ein Zeichen einfügen (INST)
$9C	Cursorfarbe purpur
$9D	Cursor eine Spalte nach links
$9E	Cursorfarbe gelb
$9F	Cursorfarbe türkis

Bild 7. Die Steuerzeichen und Ihre Hex-Codes

```
Befehl:      T ("Transfer")
Syntax:      T Anfangsadr. Endadr. Zieladr.
Wirkung:     verschiebt den Bereich von
             Anfangsadr.
             bis Endadr. an die Zieladr.
Beispiel:    T 8000 8300 C00 verschiebt einen Teil
             des ROMs in den Bildschirmspeicher;
             am Bildschirm erscheint ein
             Zeichengewirr.
```

Bild 8. Kurzübersicht zum Befehl »T«

```
Befehl:      C ("Compare")
Syntax:      C Anfangsadr. Endadr. Vergleichsadr.
Wirkung:     vergleicht den Bereich von
             Anfangsadr. bis Endadr. mit dem
             Bereich ab Vergleichsadr.;
             unterschiedliche Adressen werden
             ausgegeben.
Beispiel:    C 80CF 80D9 E3A7
```

Bild 9. Kurzübersicht zum C-Befehl

```
100 REM **********************
110 REM *                    *
120 REM *  ASCII - TABELLE   *
130 REM *                    *
140 REM **********************
150 REM *                    *
160 REM *  FLORIAN MUELLER   *
170 REM *                    *
180 REM **********************
190 :
200 DO : INPUT "DRUCKER (D) ODER BILDSCHIRM (B)"
    ;A$: LOOP WHILE A$<>"D" AND A$<>"B"
210 IF A$="B" THEN 250
220 INPUT "GERAETENUMMER";G: INPUT "SEKUNDAERADR
    ESSE";S
230 OPEN 4,G,S: CMD 4
240 :
250 LE$=CHR$(32): REM LEERZEICHEN
260 Z$="0123456789ABCDEF": REM ZIFFERN FUER HEX-
    ZAHLEN
270 PRINT SPC(2);
280 :
290 FOR J = 1 TO 16
300 : PRINT MID$(Z$,J,1);LE$;
310 NEXT J
320 PRINT
330 :
340 FOR J = 0 TO 15
350 : PRINT MID$(Z$,J+1,1);SPC(5)
360 : FOR K = 2 TO 15: IF K>7 AND K<10 THEN 390
370 : : PRINT CHR$(J+(K*16));LE$;
380 : : IF K=7 THEN PRINT LE$;LE$;LE$;LE$;
390 : NEXT K
400 : PRINT
410 NEXT J
420 :
430 IF A$="D" THEN PRINT# 4: CLOSE 4
```

Listing 1. Erstellen einer ASCII-Tabelle

gibt diesen Suchpuffer aus, wo wir die ASCII-Codes des Textes dem Memory Dump entnehmen können:

```
M 0 25D
```

Diese Art von Umrechnung ist immer noch die beste, schnellste und komfortabelste.

Zurück zu den Basic-Befehlen für ASCII-Codes. Mit folgendem Befehl läßt man das zu einem ASCII-Code gehörende Zeichen ausgeben:

```
PRINT CHR$(DEC("hex.ASCII-Code"))
```

Mit <PRINT CHR$(DEC("41"))> erhalten wir zum Beispiel das Zeichen »A«.

Probieren Sie aber doch einmal die ASCII-Codes $13, $93, $1F, $90 oder $05 aus. Wie Sie sehen, ergeben diese kein Zeichen, sondern positionieren den Cursor, ändern die Zeichenfarbe oder ähnliches. Dies sind sogenannte Steuerzeichen. In Bild 7 finden Sie alle C 16-Steuerzeichen mit ihren Commodore-ASCII-Codes.

Ein interessantes Steuerzeichen ist »ESC« (27); dazu lesen Sie bitte im Handbuch nach.

Wir wollen nun weitere Monitor-Kommandos kennenlernen, die einfach zu handhaben sind. Alle wichtigen Informationen über die Syntax und so weiter finden Sie in den Kurzübersichten.

Der Befehl <T> dient zum Verschieben von Speicherbereichen (Bild 8). Dieser Verschiebevorgang wird fortlaufend überprüft und Adressen, an denen das Verschieben nicht ordnungsgemäß durchgeführt werden konnte, werden ausgegeben.

Der Vergleich von Speicherbereichen ist mit <C> möglich (Bild 9), ganze Speicherbereiche mit einem Wert anfüllen kann man über <F> (Bild 10).

Um Speicherbereiche zu laden, speichern oder zu verifi-



S. Baloui
C64-Fischertechnik Messen, Steuern, Regeln
Februar 1986, 174 Seiten

Dieses Buch bietet einen ausführlichen Programmierkurs für die Entwicklung von Steuerungssoftware mit dem Fischer-Computing-Baukasten.

Best.-Nr. MT 90194
ISBN 3-89090-194-8 **DM 29,90**



H. Haberl
Mini-CAD mit Hi-Eddi plus auf dem C64
Januar 1986, 230 Seiten inkl. Disk

Das Zeichenprogramm »Hi-Eddi« aus der Zeitschrift 64'er fand so großen Anklang bei den Lesern, daß sich sein Schöpfer H. Haberl angespornt sah, dieses erfolgreiche Programm zu einem umfangreichen und leistungsfähigen CAD-Programm für den C64 auszubauen. Auf der beiliegenden Diskette findet der Leser das vollständige Programm, mit dem das komfortable Erstellen von technischen Zeichnungen, Plänen oder Diagrammen ebenso möglich ist wie das Malen von farbigen Bildern, Entwurf und Ausdruck von Glückwunschkarten, Schildern, ja sogar von bewegten Sequenzen (kleine Trickfilme, Schaufenster-Werbung).

- Wer sagt, daß CAD auf dem C64 nicht möglich ist?!

Best.-Nr. MT 90136
ISBN 3-89090-136-0 **DM 48,-**

Markt & Technik-Fachbücher erhalten Sie bei Ihrem Buchhändler.

Markt&Technik
Unternehmensbereich Buchverlag
Hans-Pinsel-Straße 2, 8013 Haar bei München

REX INH. ANDREAS KÖNIG **5800 HAGEN 1**
DATENTECHNIK **STRESEMANNSTR. 11**
TEL. 02331/32734-5 + 16979, TLX. 823401
Alles für den C 64/128

WELTNEUHEIT
1 MEGABYTE (1.000.240,00 BYTE) EPROMKARTE
Die GOLIATH-KARTE für C 64/128

Die Sensation ist perfekt. Eine Epromkarte für den Expansionsport des 64'er und 128'er. Ebenfalls auch für den neuen 64'er und 128D.

Steckbrief:
★ mit eingebautem Steuereprom ★ 16 Steckplätze für je 2764/128/256/512 ★ wahlweise über eigene Directory oder F-fastensteuerung ★ für Erweiterung wie Prologidos oder ähnlich ★ integrierter Modulmanager ★ vollständig gebufferter Bus ★ Hard- und Software-mäßig schaltbar ★ umschaltbar von IO1 auf IO2 ★ Expansionsport durchgeführt ★ eingebauter Resettaster und Stützfüße ★ ansonsten sind die Daten wie die 256-K-Epromkarte, welche jetzt zum Sonderpreis von 109,– DM verkauft wird. Siehe Best.-Nr. 9513 ★ Alle Programme, die mit dem Modulmanager der 256-K-Karte erstellt wurden, laufen ebenfalls auf der 1-MB-Karte!

Best.-Nr. 9600 Fertiggerät 149,50

DER GOLIATH-EPROMER
64-K-EPROMER mit PFIFF

★ brennt »fast« alles, was ihm in den Textool kommt.
★ brennt 1 KB in 1 Sekunde
★ 3 prg. Algorhythmen zur Auswahl
★ Einsteckleiste für Steckmodule zum direkten Einlesen!

Für folgende Eproms:
★ 2508/16/32/64 ★ 2716/58/32/64/128/256/512/513 ★ 87c56/5133/5143/27916/x12816a/x12864a/x148c64 ★ liest : 2332/2364 (ROMs) ★ und noch vieles mehr.

Nr. 9610 Fertiggerät 298,–

Der Epromer 32 K

Dieser Spitzenromer im Gehäuse ist eine weitere Bestleistung aus unserem Hause. Er brennt alle Eproms 2716 bis 27256 sowie die CMOS-Eproms, welche pinkompatibel sind. Die Programmierspannungen von 12,5/21/25 Volt sind softwaregesteuert. Der Anschluß erfolgt direkt ohne jeden Zusatz am Userport!

Technischer Steckbrief:
- Editor/Monitor
- umschaltbar: Hexa- und Dezimal
- Auslesen/Leertest/Brennen
- ständiger Speichervergleich
- Einzelbyte-/Schnellprogrammierung
- Standardprogrammierung/Wiederholen
- Autostartgenerator für Masch.-Basicptg.
- eingebauter Modulgenerator
- komfortable deutsche Menüführung

Lieferumfang:
- Komfortable Software auf Diskette
- auf Wunsch die Software auf Eprom
- 10seitige deutsche Beschreibung
- Datenblatt von Eproms
- ein Leereprom zum Testen

9526 Bausatz ohne Textoolsockel
9534 Bausatz mit Textoolsockel
9511 Fertiggerät ohne Textoolsockel ohne Gehäuse
9555 Fertiggerät mit Textoolsockel mit Gehäuse
9556 Auf Wunsch liefern wir die Softw. auf Eprom
9557 Ebenfalls liefern wir das Leergehäuse einzeln

256-K-Epromkarte der Superlative

Der absolute RENNER unter den Hardwareangeboten. Seine Speicherkapazität übertrifft die einer Diskette. Echte **256 K** »256.000« Bytes mehr.

Dazu kommen die Merkmale wie:
- Kein lästiges Laden mehr
- jedes Byte wird genutzt
- keines verschenkt
- eingebauter Modulmanager
- Erstellen eigener Directory

Der MODULMANAGER ermöglicht es, lückenlos Basic- oder Maschinenprogramme brennfertig und lauffähig zu machen.

Beispiel: Sie können 256 Programme mit 1 K, auf die Karte »packen«, oder aber auch 1 Programm mit 250 K!!!

Technischer Steckbrief: 8 freie Steckplätze für je 8/16 oder 32 K oder besser gesagt für 2764/27128 oder 27256. Natürlich kann auch gemischt werden.

Der Anschluß erfolgt direkt am Modulport. Die deutsche Menüführung ist äußerst komfortabel. Ein Eingriff in den Rechner ist nicht erforderlich. Dadurch bleibt ein evtl. Garantieanspruch erhalten.
Die Karte hat etwa Europaformat (inkl. Stecker), also ca. 100 x 160 mm. Füße zum Abstützen der Karte werden mitgeliefert.

9574 Komplettbausatz
9513 Fertiggerät

Eine gute Sache ist der Winkeladapter. Dieses kleine, doch »oft sehr wichtige« Zubehör ermöglicht den Betrieb der oben genannten Karte um 96° versetzt, also für den Senkrechtbetrieb. Für alle, die hinter dem Rechner nicht viel Platz haben. Der Adapter wird fertig mit Füßen geliefert. Außerdem ist der Modulport nach hinten hin noch für weitere Anschlüsse frei.

9575 Winkeladapter komplett
9402 Leerplatine f. W-Adapter

Geschäftsbedingungen:
Lieferung schnellstens per Nachnahme oder Vorkasse. Ab 200,– Warenwert frachtfrei. Versicherung 1,– Meist erfolgt die Lieferung am gleichen Tag, wenn der Auftrag bis 13 Uhr eingeht. Vierfarb-Katalog gegen –,80 DM in Briefmarken. Bei Bestellungen wird der Katalog kostenlos beigelegt.
Händleranfragen erwünscht!

VERSAND:
REX-DATENTECHNIK-HAGEN
Andreas König, Stresemannstraße 11, 5800 HAGEN 1
Telefon 02331/16979 + 32734, Telex 823401
Postgirokonto Dortmund: Andreas König 16873-467

zieren, gibt es die Befehle <S>, <L> und <V> (Bilder 11 bis 13), die den Basic-Befehlen »LOAD«, »SAVE« und »VERIFY« entsprechen, sich aber nicht auf das im Speicher befindliche Basic-Programm, sondern auf einen Teil des Speichers, den Sie selbst bestimmen, beziehen.

Bei <S> wird der Inhalt der Endadresse nicht mitgespeichert; »S "NAME",1000,2000« speichert also von $1000 bis $1FFF, nicht bis $2000. Gegebenenfalls müssen Sie also die Endadresse um 1 erhöhen.

Bearbeiten Sie ein Basic-Programm, ist es besser, die entsprechenden Basic-Befehle (»LOAD«, »SAVE«, »VERIFY«) heranzuziehen. Sie können aber auch die Monitor-Kommandos verwenden, wenn Sie Anfangs- und Endadresse des Basic-Programms kennen. Die Monitor-Befehle sind lediglich dann unausweichlich, wenn Sie nur bestimmte Teile des Speichers speichern oder laden wollen.

Wir wollen uns noch ein wenig mit der Ein-/Ausgabe des TEDMON befassen. Zunächst klären wir die beiden unterschiedlichen Ladevorgänge. Wenn Sie ein Programm über <L "NAME", Gerät> laden, wird es an die Adresse, ab der es gespeichert wurde, geladen.

Wenn ein Programm gespeichert wird, kommen auf das Speichermedium nicht nur die Daten des Programms, sondern auch die Anfangsadresse. Auf Diskette geben die ersten zwei Bytes des Programms die Anfangsadresse an (1. Byte: Low-Byte; 2. Byte: High-Byte).

Listing 2 (»FILE-INFO«) ermöglicht Ihnen, Anfangs- und Endadresse eines Programms zu ermitteln, das auf einer Diskette gespeichert ist.

Nach Eingabe des Filenamens (<$> bewirkt Ausgabe des Disk-Inhaltsverzeichnisses) stellt das Programm die Anfangsadresse und die Anzahl der Diskettenblöcke fest. Daraus läßt sich (aus der Kapazität eines Diskettenblocks) die maximale Endadresse des untersuchten Programms ermitteln. Die genaue Endadresse wird in einem etwas längeren Vorgang festgestellt; diese können Sie dann direkt als Parameter für den S-Befehl einsetzen. Am besten erstellen Sie sich einen Katalog aller Programme, die Sie mit dem Monitor bearbeiten wollen, wobei die optionale Druckerausgabe sicher eine Hilfe ist.

Insbesondere, wenn Sie später mit Maschinenprogrammen arbeiten wollen, werden Sie auf Listing 2, das »FILE-INFO«, nicht verzichten wollen.

TEDMON und hochauflösende Grafik

Eine Anwendung des S-Befehls ist das Speichern der hochauflösenden Grafik.

Mit

```
S "GRAFIK",Gerät,2000,4000
```

können Sie die hochauflösende Grafik speichern.

Mit

```
L "GRAFIK",Gerät
```

wird sie wieder geladen.

Druckerausgabe mit dem TEDMON

Leider bietet der TEDMON keinen Befehl, um die Ausgabe vom Bildschirm auf den Drucker umzulenken. Mit einem Basic-Befehl ist das aber möglich. Geben Sie in Basic

```
OPEN 4,4:CMD 4:MONITOR
```

ein, wird vor dem Start des Monitors die Ausgabe auf den Drucker umgelenkt. Dies geht soweit, daß sogar die Einschaltmeldung des TEDMON auf den Drucker geleitet wird. Sie geben aber ungeachtet dessen die Monitor-Kommandos wie üblich ein. Alle Ausgaben des Monitors (Memory Dumps etc.) kommen nun auf den Drucker.

Wenn Sie fertig sind, genügt es nicht, beide Geräte einfach auszuschalten, sondern Sie müssen den Monitor über <X> verlassen und in Basic dann

Befehl:	F ("Fill")
Syntax:	F Anfangsadr. Endadr. Füllbyte
Wirkung:	füllt den Bereich von Anfangsadr. bis Endadr. mit dem Füllbyte
Beispiel:	F C00 F00 01 füllt den Bildschirm(speicher) mit "A"

Bild 10. Kurzübersicht zum Befehl »F«

Befehl:	S ("Save")
Syntax:	S "Filename",Gerätenummer, Anfangsadr. Endadr.
Wirkung:	unter dem Filenamen in Anführungszeichen wird der Bereich von Anfangsadr. bis Endadr. auf das Gerät, das durch "Gerätenummer" angewählt wird, abgespeichert.

Bild 11. Kurzübersicht zum Befehl »S«

```
100 REM ********************************
110 REM *                              *
120 REM *        ANFANGSADRESSE        *
130 REM *                              *
140 REM *        UND ENDADRESSE        *
150 REM *                              *
160 REM *   EINES DISKETTENPROGRAMMS   *
170 REM *                              *
180 REM *          ERMITTELN.          *
190 REM *                              *
200 REM ********************************
210 REM *                              *
220 REM *  1985/86 BY FLORIAN MUELLER  *
230 REM *                              *
240 REM ********************************
250 :
260 SCNCLR : W$="#####"
270 PRINT CHR$(27)"O": PRINT "BESTIMMUNG VON ANF
    ANGS- UND ENDADRESSE" CHR$(13)
280 INPUT "FILENAME";N$: IF N$="$" THEN PRINT :
    DIRECTORY : PRINT : GOTO 270
290 OPEN 15,8,15,"I0"
300 IF DS<>0 THEN 330
310 OPEN 1,8,3,N$+",P,R"
320 IF DS=0 THEN 400
330 PRINT : PRINT CHR$(15)"DISKFEHLER: ";DS$: PR
    INT
340 CLOSE 1: CLOSE 15: GOTO 280
350 :
360 REM ************************
370 REM * INFORMATIONEN AUSGEBEN *
380 REM ************************
390 :
400 SCNCLR : PRINT : PRINT "INFORMATIONEN FUER "
     CHR$(34);N$;CHR$(34)
410 PRINT
420 :
430 REM ******************
440 REM * ANFANGSADRESSE *
450 REM ******************
460 :
470 PRINT : PRINT "ANFANGSADRESSE:",
480 GET #1,A$: GET #1,B$
490 IF A$="" THEN A$=CHR$(0)
500 IF B$="" THEN B$=CHR$(0)
510 A=ASC(A$)+256*ASC(B$)
520 PRINT USING W$;A;: PRINT " = $";HEX$(A)
530 CLOSE 1
540 :
550 REM **********************
560 REM * ANZAHL DER BLOECKE *
570 REM **********************
580 :
590 PRINT : PRINT "DISKBLOECKE:",
600 OPEN 1,8,0,"$:"+N$
610 GET #1,A$,B$
620 GET #1,A$,B$
630 GET #1,A$,B$
640 GET #1,B$: IF ST<>0 THEN 830
650 IF B$<>CHR$(34) THEN 640
660 DO : GET #1,B$: LOOP WHILE B$<>CHR$(34)
670 DO : GET #1,B$: LOOP WHILE B$=CHR$(32)
```

```
Befehl:       L ("Load")
Syntax:       L "Filename",Gerät,Ladeadresse
        oder  L "Filename",Gerät
        oder  L "Filename"
Wirkung:      Das Programm »Filename« wird vom Gerät
              an die Ladeadresse geladen. Fällt die
              Ladeadresse weg, wird das Programm in
              den Bereich geladen, aus dem es auch
              abgespeichert wurde. Gibt man das
              Gerät nicht an, wird $01 (Datasette)
              angenommen.
              Durch die Angabe einer Ladeadresse
              kann man das Programm beim Laden in
              einen anderen Speicherbereich
              schreiben als den, in dem es vor dem
              Speichern stand.
```

Bild 12. Kurzübersicht zum L-Befehl

```
680 DO : GET #1,B$: LOOP WHILE B$<>""
690 GET #1,A$,B$,A$,B$
700 IF A$="" THEN A$=CHR$(0)
710 IF B$="" THEN B$=CHR$(0)
720 B=ASC(A$)+256*ASC(B$)
730 PRINT USING W$;B;: PRINT " = $";HEX$(B)
740 CLOSE 1
750 :
760 REM ***********************
770 REM * MAXIMALE ENDADRESSE *
780 REM ***********************
790 :
800 PRINT : PRINT "MAX.ENDADRESSE:",
810 E=A+254*B-2
820 PRINT USING W$;E;: PRINT " = $";HEX$(E)
830 CLOSE 1: CLOSE 15
840 PRINT : PRINT CHR$(18)"TASTE DRUECKEN"
850 PRINT : PRINT "(G = GENAUE ENDADR./ D = DRUC
    KERAUSGABE)";
860 GET KEY GK$: IF GK$="D" THEN EE=E: M=1: GOTO
     1070: ELSE IF GK$<>"G" THEN RUN
870 :
880 REM ********************
890 REM * GENAUE ENDADRESSE *
900 REM ********************
910 :
920 M=0
930 PRINT CHR$(27)"D";CHR$(145) CHR$(145)"ENDADR
    ESSE FUER 'S':";
940 EE=A: OPEN 1,8,0,N$
950 DO UNTIL ST AND 64
960 : GET #1,A$: EE=EE+1
970 LOOP : EE=EE-1
980 PRINT USING W$;EE;: PRINT " = $";HEX$(EE)
990 CLOSE 1
1000 PRINT : PRINT CHR$(18)"TASTE DRUECKEN (D =
     DRUCKERAUSGABE)";CHR$(146);
1010 GET KEY GK$: IF GK$<>"D" THEN RUN
1020 :
1030 REM ******************
1040 REM * DRUCKERAUSGABE *
1050 REM ******************
1060 :
1070 V$="################################": REM 30
      * #
1080 PRINT CHR$(27)"D"
1090 OPEN 4,3
1100 PRINT#4,USING V$;"FILENAME:";: PRINT#4,N$
1110 PRINT#4,USING V$;"ANFANGSADRESSE:";: PRINT#
     4,USING W$;A;
1120 PRINT#4," = $";HEX$(A)
1130 PRINT#4,USING V$;"ENDADRESSE:";: PRINT#4,US
     ING W$;E;: PRINT#4," = $";HEX$(EE);
1140 IF M=1 THEN PRINT#4," (MAXIMAL)": ELSE PRIN
     T#4," (FUER 'S'-BEFEHL)"
1150 PRINT#4,USING V$;"LAENGE IN DISK-BLOECKEN:"
     ;: PRINT#4,USING W$;B;
1160 PRINT#4," = $";HEX$(B)
1170 CLOSE 4
1180 IF M=1 THEN 840: ELSE GOTO 1000
```

Listing 2. »File-Info« für C16, C116 und Plus/4

```
PRINT#4:CLOSE4
```

eingeben. Wenn Sie das unterlassen, ist es gut möglich, daß der Drucker eine auszugebende Zeile nicht aufs Papier bringt, sondern »verschluckt«.

Jetzt haben wir alle Befehle, die zum Bearbeiten von Basic-Programmen mit dem Monitor erforderlich sind, besprochen. Manche Befehle werden wir für unsere Anwendungen auf Basic-Programme, zu denen wir nun kommen, noch gar nicht benötigen. Ich habe diese Kommandos dennoch vorgestellt, denn Sie könnten durchaus in die Lage kommen, auf diese zurückgreifen zu müssen.

Basic-Programme durchleuchten

Als eine der ersten Anwendungen des M-Befehls haben wir uns ein Basic-Programm angesehen. Dabei blieben noch viele Fragen offen (zum Beispiel warum man kein Befehlswort wie »PRINT« oder »GOTO« sehen konnte), die nun nach und nach beantwortet werden können. Dazu analysieren wir das Memory Dump eines Beispiel-Programms, an dem wir später auch Manipulationen durchführen wollen. Dieses Beispiel-Programm ist Listing 3. Das Memory Dump lassen Sie nun wie beim M-Befehl besprochen ausgeben (Anwendung 2 des M-Befehls).

Sie erhalten dann den Ausdruck, den Sie auch in Bild 14 sehen können.

Listing 3 müssen Sie übrigens exakt so eintippen, wie es ausgedruckt ist; sie dürfen (außer direkt nach der Zeilennummer) kein Leerzeichen auslassen!

```
100 REM "PROGRAMM VERSUCHSKANINCHEN"
110 SCNCLR : PRINT "(C) 64'ER"
120 REM ******************************
130 COLOR 4,1
140 END
```

Listing 3. Kleines »Versuchskaninchen«

Wir besprechen nun anhand des »Versuchskaninchen«-Programms den Aufbau eines Basic-Programms im Speicher.

Die Linkpointer

Die ersten Bytes des Programms enthalten den sogenannten »Linkpointer« der ersten Programmzeile. Der Begriff »Linkpointer« heißt auf deutsch etwa »Verbindungszeiger«. Der Computer benötigt den Linkpointer, um zu wissen, wo im Speicher die nächste Programmzeile steht. Am Anfang jeder Zeile steht im Speicher der Linkpointer auf die nächste. In unserem Fall - der Linkpointer ist immer im Low-High-Format abgelegt - beginnt also die nächste Zeile bei $1024, denn am Anfang steht »24 10«.

Die Zeilennummer

Der Computer muß sich auch die Zeilennummer merken. Diese legt er ebenfalls im Low-High-Format ab. Die Zeilennummer steht unmittelbar nach dem Linkpointer, also im 3. und 4. Byte jeder Zeile. In unserem Fall ist die Zeilennummer durch die Bytes »64 00« bestimmt ($0064 = 100).

Die Token

Nach der Zeilennummer finden wir im Listing des Programms ein Leerzeichen. Das nächste Byte im Speicher (»8F«, siehe Memory Dump) ist aber kein Leerzeichen.

Beim LISTen eines Programms fügt der Computer automatisch nach der Zeilennummer genau ein Leerzeichen ein; bei der Eingabe aber ignoriert er alle Leerzeichen zwischen Zeilennummer und Befehl. Es ist also völlig egal, ob Sie

```
100      REM "PROGRAMM VERSUCHSKANINCHEN"
```

oder

```
100REM"PROGRAMM VERSUCHSKANINCHEN"
```

eingeben.

Später gibt er der Übersichtlichkeit halber von sich aus eines nach der Zeilennummer aus.

Im Speicher ist also das nächste Byte nicht das Leerzeichen, sondern der REM-Befehl.

Nun finden wir aber nach der Zeilennummer nicht die ASCII-Codes des Textes »REM«, sondern nach der Zeilennummer (»64 00«) nur das Byte »8F«).

Der Grund ist, daß Befehlswörter nicht im ASCII-Format gespeichert werden, sondern schon bei der Eingabe eine Codierung in Token erfolgt. Jedes Befehlswort wird durch ein Byte (im Beispiel »8F« für REM) repräsentiert. Dies spart Speicherplatz und erhöht die Verarbeitungsgeschwindigkeit, weil jetzt anstatt mehrerer Bytes nur 1 Byte ($8F) vom Computer bearbeitet werden muß.

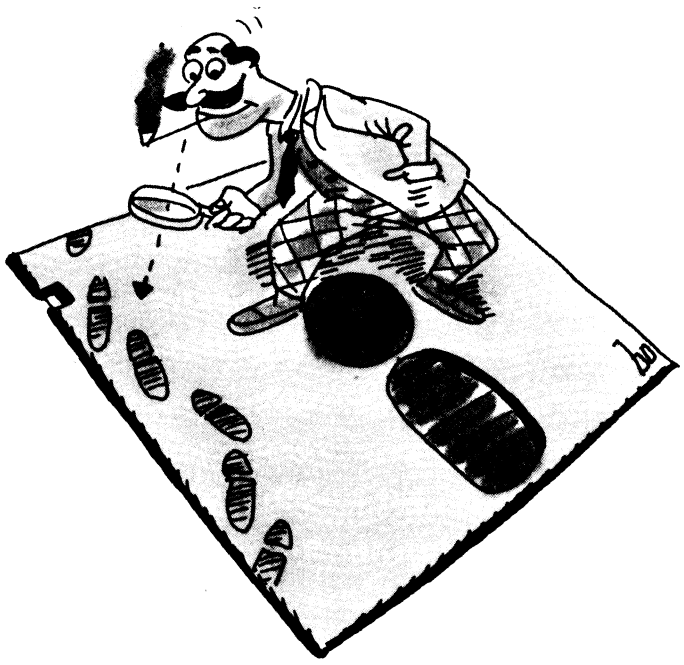
Token erkennt der Computer daran, daß sie größer als $80 sind.

Es ist noch anzumerken, daß der Computer nur Befehlswörter außerhalb von Anführungszeichen in Token umwandelt.

Die Endmarkierung einer Zeile

Nach dem REM-Token (»8F«) finden wir die ASCII-Codes des Textes nach REM. Im ASCII-Code merkt sich der Computer einen Großteil des Programms, nämlich:

- Texte in Anführungszeichen
- Texte nach einem REM-Befehl
- Texte, die nicht Befehlswörter (also zum Beispiel Variablen) sind



In der Dump-Zeile mit der Basisadresse $1021 steht am Anfang »4E 22«. Das sind die ASCII-Codes für »N« und das Anführungszeichen »"«, also der Schluß des Textes nach dem REM-Befehl in Zeile 100. Da an dieser Stelle die Zeile 100 aufhört, folgt eine Endmarkierung in Form eines $00-Bytes. Am Ende jeder Zeile steht $00. Darauf folgt dann die nächste Zeile in der gewohnten Reihenfolge »Linkpointer – Zeilennummer – Befehl – Endmarkierung«. Nach der letzten Zeile steht dann ein $00-Byte als Endmarkierung der Zeile ($1068).

Dann folgen aber noch zwei weitere $00-Bytes ($1069/$1070), die als Endmarkierung des gesamten Programms dienen. Im Memory Dump stehen nach $1070 zwar

```
Befehl:         V ("Verify")
Syntax und Wirkung siehe L-Befehl (Bild 12); bei V wird
das Programm nicht geladen, sondern wie beim Befehl
"VERIFY" in Basic verglichen.
```

Bild 13. Kurzübersicht zum V-Befehl

```
>1001 24 10 64 00 8F 20 22 14  :
>1009 0D 93 12 4B 4F 50 46 5A  :
>1011 45 49 4C 45 92 20 2A 2A  :
>1019 2A 2A 0D 0D 1B 54 2A 2A  :
>1021 2A 8D 00 37 10 6E 00 E8  :
>1029 3A 99 22 28 43 29 20 36  :
>1031 34 27 45 52 22 00 5A 10  :
>1039 78 00 8F 20 1B 44 45 53  :
>1041 43 2D 44 20 4C 4F 45 53  :
>1049 43 48 54 20 44 49 45 20  :
>1051 5A 45 49 4C 45 2E 2E 2E  :
>1059 00 63 10 82 00 E7 34 2C  :
>1061 31 00 69 10 8C 00 80 00  :
>1069 00 00 00 00 00 00 00 00  :
```

Bild 14. Memory Dump des Programms »Versuchskaninchen«

```
>1001 24 10 64 00 8F 20 22 50  :
>1009 52 4F 47 52 41 4D 4D 20  :
>1011 56 45 52 53 55 43 48 53  :
>1019 4B 41 4E 49 4E 43 48 45  :
>1021 4E 22 00 37 10 6E 00 E8  :
>1029 3A 99 22 28 43 29 20 36  :
>1031 34 27 45 52 22 00 5A 10  :
>1039 78 00 8F 20 2A 2A 2A 2A  :
>1041 2A 2A 2A 2A 2A 2A 2A 2A  :
>1049 2A 2A 2A 2A 2A 2A 2A 2A  :
>1051 2A 2A 2A 2A 2A 2A 2A 2A  :
>1059 00 63 10 82 00 E7 34 2C  :
>1061 31 00 69 10 8C 00 80 00  :
>1069 00 00 00 00 00 00 00 00  :
```

Bild 15. »Versuchskaninchen« mit Steuercodes

```
>1001 24 10 64 00 8F 0D 12 50  :
>1009 52 4F 47 52 41 4D 4D 20  :
>1011 56 45 52 53 55 43 48 53  :
>1019 4B 41 4E 49 4E 43 48 45  :
>1021 4E 0D 00 37 10 6E 00 E8  :
>1029 3A 99 22 28 43 29 20 36  :
>1031 34 27 45 52 22 00 5A 10  :
>1039 78 00 8F 20 1F 42 4C 41  :
>1041 55 13 55 4E 44 20 57 49  :
>1049 45 44 45 52 20 4C 49 4E  :
>1051 4B 53 20 4F 42 45 4E 2E  :
>1059 00 63 10 82 00 E7 34 2C  :
>1061 31 00 69 10 8C 00 80 00  :
>1069 00 00 00 00 00 00 00 00  :
```

Bild 16. Programm mit Codes, die größer als $7F sind und mit ESC-Befehlen

noch mehrere Nullen, diese gehören aber nicht mehr zum Programm, sondern stehen nur zufällig dort.

Nachdem wir nun den Aufbau besprochen haben, sollten Sie in der Lage sein, jedes einzelne Byte im Memory Dump des Beispielprogramms zu erklären; die Token müssen Sie natürlich nicht auswendig wissen.

Wir wenden uns den ersten Manipulationen an unserem »Versuchskaninchen« zu. Speichern Sie bitte spätestens jetzt dieses Programm.

Linkpointer-Manipulationen

Die Linkpointer werden vom Computer nach jedem Laden eines Programms neu berechnet; deshalb kann man zwar die Linkpointer trickreich manipulieren und das Programm abspeichern, nach dem Neuladen aber ist die ganze Mühe umsonst. Aus diesem Grund sind die Effekte, die wir nun hervorrufen werden, lediglich eine Demonstration, von einem echten LIST-Schutz kann man nicht sprechen.

Als erstes wollen wir den Linkpointer von Zeile 100 auf sich selbst stellen, so daß beim LISTen nur immer Zeile 100 gelistet wird, der Rest des Programms aber nicht.

Der erste Linkpointer steht bei $1001 und zeigt noch auf $1024. Geben Sie <M 1001> ein und ändern Sie die beiden ersten Bytes auf <01 10> ($01: Low-Byte von $1001, $10: High-Byte), oder geben Sie einfach <>1001 01 10> ein.

»Vermurkste« Basic-Programme

Um das Zerstörungswerk anzusehen, verlassen wir den Monitor und geben <LIST> ein. Die Zeile 100 wird ununterbrochen gelistet, bis wir <RUN/STOP> drücken.

Sicherlich ist dies eine nette LISTschutz-Variante, doch können wir sie nicht einsetzen, da beim Neuladen eines Programms von Diskette oder Datasette derartige LISTschutz-Vorrichtungen entfernt werden. Der Computer bedient sich dabei einer Routine, die Sie über

```
SYS DEC("8818")
```

aufrufen können. Nach diesem Befehl ist das Programm wieder »repariert«, da die Linkpointer neu berechnet wurden.

Das soll uns aber nicht davon abhalten, weiter mit den Linkpointern zu experimentieren. Als nächstes wollen wir die Endmarkierung des Programms so ändern, daß nach dem LISTen der letzten Zeile wieder die erste Zeile geLISTet wird.

Der Linkpointer der letzten Zeile steht bei $1063 und zeigt auf die Endmarkierung ($1069). Wir ändern also das »69 10« in »01 10«, und schon haben wir ein weiteres Endlos-Listing, bei dem allerdings jede Programmzeile gelistet wird. Auch hier hilft <RUN/STOP> und dann ein

```
SYS DEC("8818")
```

weiter.

Der letzte Linkpointer-Trick ist, daß wir die Linkpointer so stellen, daß bestimmte Zeilen für den LIST-Befehl nicht vorhanden sind. Solche Zeilen dürfen aber auch nicht angesprungen werden.

Folgendermaßen sind die Linkpointer in »Versuchskaninchen« gestellt:

$1001 (Zeile 100) zeigt auf $1024,
$1024 (Zeile 110) zeigt auf $1037,
$1037 (Zeile 120) zeigt auf $105A,
$105A (Zeile 130) zeigt auf $1063,
$1063 (Zeile 140) zeigt auf $1069,
bei $1069 stehen Nullbytes als Endmarkierung.

Wir richten den ersten Linkpointer auf den letzten:

```
>1001 63 10
```

Damit wird auf $1063 (Zeile 140) gestellt.

Die Eingabe von <LIST> zeigt, daß die Zeilen 110,120 und 130 beim LISTen verschluckt werden, obwohl sie noch vorhanden sind, wie <SYS DEC("8818")> beweist.

Zugegeben: Ein wenig frustrierend ist es schon, wenn man so raffinierte Sachen mit den Linkpointern anstellt, und dann durch Speichern und Neuladen, beziehungsweise <SYS DEC("8818")> alles wieder weg ist.

Die Manipulationen, die wir nun durchführen werden, sind aber wesentlich »wetterfester«, denn man kann sie der Nachwelt durch Speichern erhalten.

Manipulationen an Zeilennummer

Wie wir wissen, kann man mit zwei Byte alle (ganzen) Zahlen von 0 bis 65535 darstellen. Dennoch läßt der Computer, der Basic-Zeilennummern ja in diesem Low-High-Format speichert, nur Zeilennummern bis 63999 zu. Mit dem Monitor sind wir aber nicht auf den normalen Basic-Eingabe-Modus angewiesen und können bestehende Zeilennummern so ändern, daß wir auch Zeilennummern von 64000 bis 65535 verwenden können.

Geben Sie folgende Monitor-Anweisungen ein (selbstverständlich ohne die dahinterstehenden Kommentare in Klammern), und schon hat unser »Versuchskaninchen« die Zeilennummern 64000 bis 64040 anstelle von 100 bis 140:

```
>1003  00 FA        (64000 statt 100)
>1026  0A FA        (64010 statt 110)
>1039  14 FA        (64020 statt 120)
>105C  1E FA        (64030 statt 130)
>1065  28 FA        (64040 statt 140)
```

Lassen Sie das Programm jetzt LISTen, Sie werden erstaunt sein!

Zwei Anmerkungen.

Erstens: Das Programm läuft nur dann korrekt, wenn Zeilen mit unerlaubt großer Zeilennummer nicht angesprungen werden, denn den Befehlen <GOTO>, <GOSUB> oder <RUN> können wir die Zeilennummern über 64000 nicht beibringen.

Zweitens: Mit einem einfachen RENUMBER-Befehl können wir auch diese Manipulation rückgängig machen; <RENUMBER 100,10> stellt den ursprünglichen Zustand wieder her.

Wir können auch allen Zeilen die Zeilennummer 100 »verpassen«:

```
>1003 64 00
>1026 64 00
>1039 64 00
>105C 64 00
>1065 64 00
```

Es gelten die gleichen Hinweise wie bei den »überdimensionalen« Zeilennummern.

Unsere nächste Anwendung ist eine echte Hilfe beim Programmieren.

Das Basic 3.5 unseres C16/116 ist zwar wirklich umfangreich, aber der FIND-Befehl zum Suchen von Texten innerhalb von Programmen fehlt leider. Mit dem Monitor können wir einen solchen unter einigermaßen vertretbarem Aufwand ersetzen.

Wir suchen jetzt als Beispiel nach dem Text »64« im Programm »Versuchskaninchen«.

I. Bestimmung von Anfangs- und Endadresse des Basic-Programms

Diesen Schritt kennen wir seit dem M-Befehl. Bei »Versuchskaninchen« ist $1001 die Anfangsadresse und $106B die Endadresse.

II. Suchen nach dem Text im Speicher

```
H 1001 106B '64
```

sucht nun nach dem Text »64«. Wir erhalten die Adresse $1030 als Ergebnis.

Befehl	Syntax	Wirkung
eXit	X	Rückkehr in Basic
Memory dump	M Start Ende	Anzeige von Speicherinhalten
>	> Adresse Byte. . .	Ändern von Speicherinhalten
Hunt	H Start Ende Byte	Suchen von Bytefolgen
Transfer	T Start Ende Ziel	Verschieben von Speicherbereichen
Copy	C Start Ende Ziel	Vergleichen von Speicherbereichen
Fill	F Start Ende Byte	Füllen von Speicherbereichen
Save	S "name", Gerät, Start, Ende+1	Speichern eines Bereichs
Verify	V "name", Gerät	Vergleichen eines Speicherbereichs

Bild 17. Tabelle der wichtigsten Befehle des TEDMON

III. Dumpen vom Anfang des Basic-Speichers bis zur angegebenen Adresse

```
M 1001 1030
```

Wir sehen uns dann von $1030 rückwärts mit ein wenig Gespür das Memory Dump an und suchen die letzte Endmarkierung. Diese ist das Null-Byte bei $1023. Bei $1026 steht dann die Zeilennummer: »6E 00«.

In Basic rechnen wir $006E mit <PRINT DEC("006E")> um und erhalten 110.

In Zeile 110 steht demzufolge der gesuchte Text.

Hat der H-Befehl in Schritt II mehrere Adressen genannt, ist Schritt III für jede Adresse durchzuführen.

LISTschutz durch ASCII-Codes von Steuerzeichen

Es gibt nicht nur für die druckenden Zeichen, sondern auch für die Steuerzeichen ASCII-Codes. In Bild 8 können Sie noch einmal nachsehen, wenn Sie Ihr Wissen auffrischen wollen.

Wir werden besonders wirkungsvolle Steuercodes in REM-Zeilen einbauen, da sie dort keinen Schaden anrichten (Texte nach REM werden nicht ausgeführt). Das Prinzip ist klar: Bei der Eingabe einer REM-Zeile werden zunächst Kommentare eingegeben, die später mit Steuercodes überschrieben werden. Beim LISTen der Zeile werden diese Steuercodes dann ausgeführt.

In Bild 15 sehen Sie ein Memory Dump einer manipulierten Form von »Versuchskaninchen«. Bitte geben Sie einmal dieses Memory Dump ein und lassen Sie dann den Basic-Befehl <LIST> ausführen, um alle Effekte zu sehen. Warum dies so ist, können Sie sich mit Hilfe der Steuerzeichentabelle selbst erklären.

Die verwendeten Steuercodes haben nur Werte bis $7F. Codes ab $80 würde der Computer als Token interpretieren und anstelle eines Steuerzeichens einen Basic-Befehl ausgeben. Dies umgeht man, indem man vor einem Steuerzeichen, das größer als $7F ist, folgende Bytes setzt:

»22 14 0D«

Pro REM-Zeile sollte man dies nur einmal machen. Dann sind alle Steuerzeichen erlaubt. Die genaue Funktionsweise dieser drei Byte wollen wir hier nicht ausbreiten, nur soviel: Der Computer wird dazu gebracht, jedes Zeichen, daß dieser Sequenz folgt, auszugeben, ohne es vorher zu decodieren.

Als Bild 16 finden Sie eine Version von Versuchskaninchen, in der dieser Trick angewendet wurde. Außerdem wird von den ESC-Sequenzen (siehe Handbuch) Gebrauch gemacht.

Wir sind nun am Ende des Kurses angelangt. Wir hoffen, es hat Ihnen Spaß gemacht, Ihren Computer an einigen Stellen zu durchschauen und »auszutricksen«.

In Bild 17 sehen Sie noch eine Tabelle mit allen Befehlen des TEDMON. Der C 128 enthält übrigens im Gegensatz zum C 64 auch einen eingebauten Maschinensprache-Monitor. Die Befehle fast aller Monitore sind von der Syntax fast identisch, so daß keine Probleme mit anderen Produkten auftreten dürften!

Sicherlich werden Sie noch Zeit brauchen, bis Sie mit dem Monitor so selbstverständlich wie mit Basic umgehen können, aber dann sind Sie bereits so weit, daß Sie mit Maschinensprache beginnen können.

(F. Müller/ks)

