INPUT64 auf der CeBIT 4. 3. bis 11. 3. Halle 7, Stand D28

DAS ELEKTRONISCHE MAGAZIN 3/87

INPUT 64

Infos · News · Programme · Unterhaltung · Tips

Vokabel-Trainer

Lernprogramm mit fertigen Dateien

BCD-Arithmetik

Rechengenauigkeit auf 250 Stellen

Neuer Kurs

6502-Maschinensprache

Mit integriertem Prozessor-Simulator

Rätsel:
Dechiffrierung

Serien:
64er Tips
Englische Grammatik

Spiele:
Happy Hacker
Selbstlernendes Tic-Tac-Toe

Dokumentation und Bedienungshinweise

INPUT-ein Magazin aus dem Verlag Heinz Heise GmbH, Postfach 61 04 07, 3000 Hannover 61

Liebe(r) 64er-Besitzer(in)!

Standardisierte Betriebssysteme sind ein — wenn nicht das — Schlagwort in der Computerterminologie. Es ist ein interessanter Aufschaukelungseffekt zu beobachten. Erst durch Setzen von Standards wird die Entwicklung von umfangreicher und kostspieliger Software möglich, da diese nicht nur für einen bestimmten Computer produziert werden muß. Durch ein großes Angebot an Software für diese Computer-Familien werden wiederum diese Rechner für die Anwender interessant. Werden mehr Rechner gekauft, gibt das wiederum einen verstärkten Anreiz für weitere Software-Entwicklung, und der Kreis ist geschlossen.

Betriebssysteme stellen elementare Routinen zur Ansteuerung der jeweiligen Hardware zur Verfügung, wobei auf der Hardwareseite in der Regel eine gewisse Anpassungsfähgkeit besteht. Für kleinere kommerzielle Rechner hat sich das Betriebssystem CP/M durchgesetzt, und bei größeren Computern (auch PCs genannt) führt an MS-/PCDOS kein Weg vorbei. Die Gründe sind sicherlich unterschiedlich. Waren gerade beim CP/M die Softwarehäuser an dieser Standardisierung interessiert, ist das Betriebssystem MS-/PCDOS von einem dominierenden Computerhersteller quasi durchgesetzt worden.

Was bedeutet nun dieses für den guten alten Commodore 64? Oberflächlich betrachtet: Der C64 ist zu nichts und niemandem kompatibel, außer zu sich selbst. Bei näherem Hinsehen: Diese ungünstige Ausgangsbasis wird durch die sehr große Verbreitung dieses Rechners mehr als kompensiert.
Die Firma Commodore meldete vor kurzem den einmillionsten verkauften C64; allein in diesem unseren Lande. Durch diese riesigen Verkaufszahlen setzt der C64 praktisch den Standard für Homecomputer schlechthin. Das haben auch die Softwarehäuser erkannt, und es gibt praktisch keine Anwendung, für die es keine fertigen Programme auf diesem Rechner gibt.

Hinzu kommt, daß weit und breit kein Computer in Sicht ist, der in absehbarer Zeit eine Konkurrenz für den C64 darstellen könnte. Sie werden also auch in Zukunft mit Ihrem C64 in keine Software-Krise geraten.

Wolfgang Rösch

INHALT

Auf einen Blick: INPUT64-Betriebssystem-Befehle

Titel abkürzen	CTRL und Q
Hilfsseite aufrufen	CTRL und H
zum Inhaltsverzeichnis	CTRL und I
Bildschirmfarbe ändern	CTRL und F
Rahenfarbe ändern	CTRL und R
Bildschirmausdruck	CTRL und B
Programm sichern	CTRL und S

Laden von Diskette:
LOAD „INPUT*",8,1
Laden von Kassette:
LOAD oder SHIFT und RUN/STOP

Ausführliche Bedienungshinweise finden Sie auf Seite 29.

Leser fragen . . .

Lohn- und Kirchensteuer

Gibt es die Möglichkeit, Ihrem Programm „Lohnsteuer 86" (INPUT 2/87) den Kirchensteuersatz für Baden-Württemberg, Bayern, Bremen und Hamburg (8 %) beizubringen?

(tel. Anfrage)

In Zeile 66 wird der Variablen KI der Wert 9 zugewiesen, entsprechend dem üblichen Steuersatz von 9 %. In Zeile 67 ist die von unserem Leser gewünschte Zuweisung von 8 % bereits vorgesehen, es muß lediglich der REM-Befehl am Anfang dieser Zeile gelöscht werden. (d. Red.)

Der Lisassembler

Ein kurzer Tip zu LISP64 (der LISP-Interpreter aus INPUT 4/86, d. Red.): Der Strukturediteor aus der Ausgabe 5/86 ist zwar abgesehen von ein paar Abstürzen recht nett, wenn man jedoch mehr visuell orientiert ist, dann bietet er halt doch nicht den gewohnten Komfort. Hierzu wäre ein Texteditor das geeignete Element, doch woher nehmen . . . Ausgabe 6/86!

Der INPUT-ASS hätte es sich zwar nicht träumen lassen, daß er einmal kein Assemblerprogramm bearbeitet, er hat aber auch nichts dagegen. Doch wie sag'ich's meinem LISP? Die beiden folgenden kurzen Funktionen erledigen das Nötige:

1.) Lesen von Diskette aus dem sequentiellen Editor-Format:

```
(DE GETDISK NIL
 (OPEN 1 8 2 "FROMEDIT,SR")
 (INPUT 1)
 (READL))
```

Diese Funktion liest ein Editor-File ein und definiert sofort die Funktionen für LISP, wenn ein,zwei Dinge bei der Erstellung beachtet werden: Am Beginn des Files muß NIL oder () (leere Klammer) stehen. Am Ende sollte zur Umschaltung auf Tastatureingaben (NORMAL) stehen. Nach dem Aufruf der Funktion mit (GETDISK) muß man sich leider noch der Mühe eines (CLOSE 1) unterziehen.

2.) Übergabe von LISP-Funktionen an den Editor:

```
(DE PUTDISK (N)
 (DISK „S:FROMLISP")
 (OPEN 1 8 2 „FROMLISP,S,W")
 (OUTPUT 1)
 (PP NIL) (*hier wird Punkt 1 beachtet, das NIL)
 (MAPC 'PDEDF N)
 (PP '(NORMAL)) (*hier wird das Ende geschrieben)
 (NORMAL)
 (CLOSE 1))
```

Das hiermit erzeugt File „FROMLISP" kann jetzt vom Editor gelesen werden. . . . Es geht doch nichts über integrierte Software aus einem Hause! (K.P. Meyer, Heilbronn)

Finden wir auch. Der Editor des Macro-Assemblers wird übrigens von vielen Lesern auch als Textverarbeitung „mißbraucht". (d. Red.)

Funktionierende Formate

Wie kann ich Zahlen so ausgeben, daß diese auf zwei Nachkommastellen gerundet sind? (tel. Anfrage)

Durch geschickte Anwendung der INT-Funktion und Dividieren und Multiplizieren. Beispielweise:

```
10 INPUT„NACHKOMMASTELLEN"; NK
20 X=2.251243124
30 AZ=INT(X*10 ↑ NK+.5)/10 ↑ NK
40 PRINT AZ
```

Eleganter geht's mit der DEF FN-Funktion:

```
10 INPUT„NACHKOMMASTELLEN"; NK
20 DEF FN R(X)=INT(X*10 ↑ NK -.5*(z>0)/10↑NK
30 X=2.231243124
40 Z=1:REM Z=0: NICHT RUNDEN
50 PRINT FN R(X)
```

(d. Red.)

SuperDisks Autostarts

Die Möglichkeit, von Ihrem Floppy-Beschleuniger SuperDisk (INPUT 1/87) die neue Benutzeroberfläche JAM (in der gleichen Ausgabe) nachladen und starten zu lassen, funktioniert bei mir nicht! (tel. Anfrage)

Damit SuperDisk auch Maschinenprogramme nachladen kann (natürlich nicht starten), wird ein eventuell nachzuladendes Programm mit der Sekundäradresse 1 geladen. Da JAM beim Abspeichern aus INPUT nicht am BASIC-Anfang liegt, wird es beim Laden durch SuperDisk nicht an den BASIC-Start geladen und kann deswegen nicht gestartet werden. Einfache Abhilfe: Das Programm JAM einmal „normal" von Diskette laden (LOAD„JAM",8,0), JAM auf der Diskette löschen und aus dem Rechner wieder abspeichern.

Außerdem muß beim Abspeichern von SuperDisk aus dem Magazin heraus darauf geachtet werden, daß wirklich „JAM" als File-Name eingetragen wird, nicht etwa „JAM " (mit abschließendem Leerzeichen)! (d. Red.)

Farbwahl bei JAM

Die Voreinstellung der Farben bei JAM (die grafisch orientierte Benutzeroberfläche aus INPUT 1/87, d. Red.) gefällt mir nicht. Gibt es eine Möglichkeit, diese dauerhaft zu ändern? (tel. Anfrage)

Folgende Werte müssen nach dem Laden und vor dem Starten von JAM in die entsprechenden Speicherstellen geschrieben werden, um die Voreinstellung der Farben zu ändern (Geänderte Version sofort abspeichern!):

Farben nach dem Start:
POKE 5566,GESCHWINDIGKEIT (1-9)
POKE 5567,RAHMENFARBE
POKE 5568,GRUNDFARBE
POKE 5569,ZEICHENFARBE
POKE 5570,PFEILFARBE

(Zur Zuordnung von Werten und Farben siehe C64-Handbuch Seite 139.)

Werte, die bei der Parametereingabe über * aufgerufen werden:

POKE 3985,ZEICHENFARBE
POKE 3996,PFEILFARBE
POKE 4004,GESCHWINDIGKEIT(1-9)

Die Farben von Rahmen und Hintergrund lassen sich hier leider nicht verändern. Frank Börncke

Wer anderen gerne Gruben hackt . . .

. . . ist bestimmt ein Freund des Happy Hacker

Dieses Spiel besteht aus insgesamt 40 Levels, diese sind in zwei Bereiche aufgeteilt. Die ersten zwanzig Level sind als Option LEVEL ← EASY im Startmenü wählbar. Anfänger sollten erst einmal in dieser Stellung üben, bevor sie die weiteren 20 Level spielen: erreicht werden diese mit LEVEL ← HARD. Um von einem Level ins nächste zu gelangen, gilt es, alle Monster, die Ihnen auflauern, unschädlich zu machen. Zuerst müssen Sie ein Monster fangen, indem Sie ihm eine Falle stellen. Hacken Sie mit der Spitzhacke einfach ein Loch; jetzt ein bißchen Geduld und gewartet, bis ein Monster in der Falle sitzt. Ist es soweit, bleibt nur noch wenig zu tun, und das Monster ist ausgeschaltet.

Hurtig zur Falle und den Pickel geschwungen! Eins, zwei, drei — ran an das nächste Viech! Der Goldsack wird natürlich nicht vergessen, denn 500 Punkte sind kein Pappenstiel, sondern eine weitere Sprosse auf dem Weg in die Bestenliste. Routinierte Monsterjäger holen sich noch einen Bonus, bevor es in den nächsten Level geht. Den gibt's allerdings nur, wenn vorher nicht getrödelt wird. Aber das werden Sie schnell mitbekommen.

Schadenfreude soll bekanntlich die reinste Freude sein, doch wer ist so böse und läßt seine Freunde in die selbstgebastelte Falle stolpern? Handelt es sich allerdings um Monster, sieht alles ganz anders aus . . .

Bei zwei Spielern können diese hintereinander spielen oder zusammen. Das Zusammenspiel bietet die Möglichkeit, auch einmal den Mitspieler auszutricksen. Wählen Sie im Menü einfach COOPERATE ← YES! Alle weiteren Fragen klären sich schnell beim ersten Probespiel. pan

Dem Prozessor auf der Spur

Die INPUT 64-Assembler-Schule

Mit unserem neuen Kurs über 6502-Maschinensprache wollen wir Sie aus dem Dunst dieser Gerüchteküche befreien. Steigen Sie ein in die 'Haute Cuisine' der Assembler-Programmierung. Im Verlauf dieses Kurses werden Sie köstliche Rezepte kennenlernen, mit denen Sie Ihren Rechner füttern können. Eine Versuchsküche, in der garantiert nichts anbrennt, liefern wir Ihnen gratis dazu.

Wenn man normalerweise anfängt, sich mit der Maschinensprache zu beschäftigen, ist spätestens nach zwei Wochen ein neuer Reset-Knopf oder gar ein neuer Netzschalter für den Rechner fällig. Ganz abgesehen von den Nerven, die durch ständiges neues Laden von Assembler und Monitor auf eine harte Probe gestellt werden.

Mit der INPUT-Assembler-Schule geht das ganz anders: In dem Programm enthalten ist in jeder Folge ein Editor, mit dem Sie eigene Maschinenprogramme erstellen können. Schon in dieser Phase werden Syntax-Fehler abgefangen. Das selbsterstellte Programm kann dann von dem ebenfalls mitgelieferten Assembler übersetzt werden.

Der eigentliche Clou ist aber der Prozessor-Simulator. Er ermöglicht das Austesten der eigenen Programme unter vollständiger Kontrolle des Rechners. Der Einzelschritt-Modus knackt selbst die hartnäckigste Endlos-Schleife, und kein undokumentierter Opcode kann den Rechner zum Absturz bringen.

Maschinensprache ist unwahrscheinlich kompliziert, schwer zu lernen, Maschinensprache-Programme sind aufwendig zu erstellen, und überhaupt ist Maschinensprache nur etwas für Profis. Diese und andere Vorurteile halten sich immer noch hartnäckig in den Köpfen vieler Computer-Besitzer. Und trotzdem träumt fast jeder davon, diese sagenumwobene Sprache zu lernen, denn überall ist zu hören, daß Maschinensprache die schnellste Programmiersprache überhaupt ist und erst sie die letzten Möglichkeiten des Computers aus ihm herausholt.

Die Flachsprache

Maschinensprache ist die 'Muttersprache' eines jeden Computers. Darum ist sie so schnell. Alle höheren Programmiersprachen wie etwa BASIC brauchen einen Übersetzer, der die Programme für den Prozessor verständlich macht. Auch im C64 läuft ein solcher Übersetzer. Es handelt sich dabei um ein Maschinensprache-Programm (was sonst?), genannt BASIC-Interpreter. Er liest ein BASIC-Programm Befehl für Befehl, entschlüsselt jeden einzelnen und führt die entsprechenden Aktionen aus.

Bei Maschinensprache-Programmen ist das ganz anders: Hier versteht der Prozessor direkt die im Speicher liegenden Befehle. Es ist wohl klar, daß so ein 'fest verdrahteter' Befehls-Ausführer schneller ist als ein Programm, das die Befehle erst in Maschinensprache übersetzen muß.

Andererseits sind die Befehle, die der Prozessor direkt versteht, nicht so leistungsfähig wie beispielsweise BASIC-Befehle. Für ein Maschinensprache-Programm muß man also das zu lösende Problem weiter 'aufdröseln'.

Ausprobieren und Simulieren

Zur Bedienung des Lernprogramms

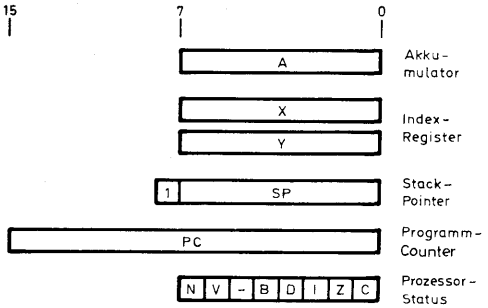
Die INPUT 64-Assembler-Schule setzt sich aus mehreren Teilen zusammen. Nach dem Laden sehen Sie ein Titelbild, von dem aus Sie mit einem beliebigen Tastendruck in das Hauptmenü gelangen.

Wenn Sie nun F1 drücken, gelangen Sie in ein Menü, das Ihnen verschiedene Themen zur Auswahl stellt. Die Erklärungen, die Sie jetzt abrufen können, sollten Sie parallel zum Beiheft lesen. Beide Medien ergänzen sich hier. Sie können die Erklärungen auch mit CTRL-b ausdrucken. Ins Hauptmenü gelangen Sie jederzeit mit der STOP-Taste zurück.

Mit F3 gelangen Sie aus dem Hauptmenü zu einer Auswahl verschiedener Beispielprogramme. Sie können eines davon auswählen, das Sie sich dann im Editor anschauen oder auch verändern können. Wenn Sie an dieser Stelle eine Null eingeben, enthält der

Binär	Dezimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Tabelle 1:
Die Zahlensysteme auf einen Blick



Bis auf den Programmzähler sind die Register des 6510 acht Bit breit. In der ersten Folge der Assembler-Schule beschäftigen wir uns mit dem Akkumulator und einem Flag im Statusregister.

Vokabeln

Wir haben gesagt, daß die Maschinensprache-Befehle so im Speicher liegen, daß der Prozessor sie direkt verstehen kann. Für uns Menschen sind sie aber nur schwer zu merken. Darum hat man die sogenannte Assembler-Sprache entwickelt. Sie besteht aus Befehlen, die sehr eng mit der Maschinensprache in Zusammenhang stehen. Jedem Maschinensprache-Befehl ist eine leicht zu merkende Kombination aus drei Buchstaben - genannt Mnemonic zugeordnet (jawohl, mit 'Mn' - ich habe das Wort nicht erfunden). Wir werden uns in diesem Kurs daher nicht mit der eigentlichen Maschinensprache auseinandersetzen, sondern mit der Assemblersprache.

Es gibt Programme, die ein in Assemblersprache geschriebenes Programm in ein Maschinen-Programm übersetzen. So ein Programm heißt Assembler. Den Text, den es verarbeitet, nennt man Source- oder Quell-Code, das Maschinen-Programm, das er erzeugt, heißt dann Object-Code. Allerdings sollte man diese strenge Begriffstrennung nicht allzu eng sehen. Man spricht oft schon von einem Assembler- oder Maschinen-Programm, wenn man eigentlich den Quell-Code meint. Oft wird auch leichthin behauptet, ein Maschinensprache-Programm sei 'in Assembler geschrieben'.

Von Bits und Bytes

Ein Bit ist die kleinste Informationseinheit eines jeden Digital-Rechners. Man kann sich ein Bit als einen Schalter vorstellen, der entweder offen ist (dann fließt kein Strom, das Bit hat den Wert Null) oder geschlossen (Stromfluß, Wert ist Eins). In solchen Schaltern kodiert jeder Computer seine Programme und Daten. Einen Code mit nur zwei Zuständen nennt man binären oder zweiwertigen Code. Der Name Bit kommt übrigens aus dem

Editor das zuletzt bearbeitete Programm, beim ersten Aufruf ist der Textspeicher leer.

Wenn Sie ein Beispielprogramm bearbeitet haben und - mit der STOP-Taste - wieder ins Hauptmenü springen, können Sie Ihren Text auch auf einen Drucker ausgeben lassen oder auf einen eigenen Datenträger abspeichern. Abgespeicherte Programme können Sie direkt mit dem INPUT-ASS (Ausgabe 6/86) laden und weiterbearbeiten.
Vom Editor aus gelangen Sie mit F7 in einen integrierten Simulator. Hier können Sie unsere Programmbeispiele oder Ihre selbstentworfenen Programme ablaufen lassen und testen, ob sie sich erwartungsgemäß verhalten.

Ausführliche Hinweise zur Bedienung des Editors und des Simulators sind im Programm enthalten. Sie können sie von dort aus jeweils mit der Funktionstaste F6 aufrufen. Es wird empfohlen, diese Seiten vor der Benutzung des Programmpakets einmal gründlich zu lesen. Besitzer eines Druckers können sie auch mit CTRL-b zu Papier bringen.

Englischen und ist die Abkürzung für Binary digiT (binäre Ziffer).

Aus mehreren binären Ziffern kann man eine Zahl bilden. Eine vierstellige Binärzahl (auch Nibble genannt) kann sechzehn verschiedene Werte annehmen. Das zeigt Tabelle 1. Auf die dritte Spalte kommen wir noch zu sprechen.

Man kann mit einem Nibble also die Zahlen Null bis Fünfzehn darstellen. Dabei hat jede Stelle einer Binärzahl den doppelten Stellenwert der rechts von ihr stehenden. In dem Ihnen wahrscheinlich geläufigeren Dezimalsystem, bei dem jede Ziffer zehn verschiedene Werte annehmen kann, hat jede Stelle ja auch den zehnfachen Wert der vorangehenden.

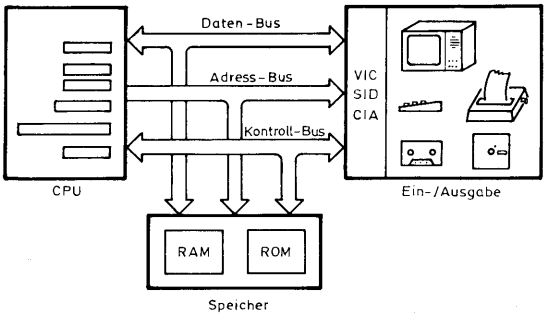
Der Prozessor, der im C64 'tickt', ist ein Acht-Bit-Prozessor. Das heißt, daß er Register (interne Speicherzellen) besitzt, die acht Bits parallel verarbeiten können. Eine solche Binärzahl mit acht Stellen heißt Byte.

Um nicht jedesmal eine achtstellige Binärzahl hinschreiben zu müssen, wenn man ein Byte meint, haben findige Mathematiker das Hexadezimalsystem erfunden. Bei ihm gibt es sechzehn verschiedene Ziffern. Ein Nibble läßt sich also mit nur einer, ein Byte mit zwei Hexadezimalziffern darstellen. Neben den gewohnten Ziffern werden noch die Buchstaben A bis F als Ziffern 'mißbraucht'. Die Zuordnung zeigt Tabelle 1.

Im Hexadezimalsystem hat jede Ziffer den sechzehnfachen Wert der vor ihr stehenden. Mit zwei Nibbles kann man also 16*16, das sind 256 verschiedene Zahlen, darstellen (0 bis 255).

Harte Sachen

Bevor es mit dem Programmieren richtig losgeht, sollen noch einige Begriffe erwähnt werden: Zum Beispiel, daß der Prozessor des C64 den Namen 6510 trägt. Er ist ein Abkömmling der 6502-CPU und versteht die gleichen Befehle wie diese. CPU ist mal wieder eine Abkürzung aus dem Englischen und kommt von 'Central Processing Unit', etwa 'Zentrale Verarbeitungseinheit'. Damit ist auch die Aufgabe eines Prozessors innerhalb eines Computersystems umrissen.



Zu einem Rechner gehören noch die beiden Funktionsblöcke Speicher und Ein-/Ausgabe-Einheit. Mit ihnen steht der Prozessor über elektrische Leitungen in Verbindung, die man Busse nennt. Es gibt einen acht Bit 'breiten' Datenbus, über den die Informationen zwischen den Komponenten ausgetauscht werden, und einen Adreß-Bus, mit dem die CPU dem Speicher mitteilt, welche Speicherstelle gemeint ist. Darüber hinaus gibt es noch Leitungen, die den zeitlichen Ablauf der Aktionen oder die Richtung des Datenflusses bestimmen. Diese Leitungen faßt man zum Kontroll- oder Steuer-Bus zusammen.

Der Adreß-Bus umfaßt beim 6510 sechzehn Leitungen. Damit können zwei hoch sechzehn, das sind 65536 verschiedene Speicherzellen angesprochen werden. Diesen Bereich teilt man in 256 Seiten (Pages) zu je 256 Adressen ein. Wir werden noch sehen, daß einige dieser Seiten für den Prozessor eine besondere Bedeutung haben.

Jedes Computersystem besteht aus drei Komponenten: einer zentralen Verarbeitungseinheit (CPU), einem Speicher und einer Ein-/Ausgabe-Einheit, Den Konakt mit der Außenwelt stellen beim C64 ein Video-Controller, ein Synthesizer-Baustein und zwei universelle Ein-/Ausgabe-Chips her.

Das erste Programm

Jetzt wollen wir aber endlich programmieren! Dabei soll unser Prozessor gleich etwas tun, was man täglich benötigt. Wir wollen ein Programm schreiben, das zwei Zahlen addiert. Der erste Assembler-Befehl, den wir benutzen wollen, heißt

LDA #6

Wie fast alles, was mit Computern zu tun hat und nicht auf Anhieb verständlich ist, handelt es sich um eine englische Abkürzung. Sie steht für 'LoaD Accumulator'. Dieser Befehl dient dazu, eine jener CPU-internen Speicherzellen, die wir Register nennen, mit einem Wert zu füllen. Dieses Register heißt Accumulator (Sammler) oder kurz Akku. Es ist ein sehr wichtiges Register, da es bei allen arithmetischen Operationen beteiligt ist.

Der Prozessor will natürlich noch wissen, welchen Wert er in den Akku laden soll. In unserem Beispiel soll das die Zahl 6 sein. Das Doppelkreuz signalisiert, daß es unmittelbar (immediate) die Zahl 6 sein soll und nicht etwa der Inhalt der Speicherstelle Nummer 6.

Als nächstes soll der Prozessor zu der im Akku gespeicherten Zahl etwas addieren, beispielsweise die Zahl 3. Der Befehl dazu lautet

ADC #3

und bedeutet 'ADd to accumulator with Carry'. Das Doppelkreuz und die 3 stellen wieder ein unmittelbares Argument dar.

Was soll nun 'with Carry' bedeuten? Dazu muß man wissen, daß es in der CPU ein sogenanntes Statusregister gibt. Eines seiner Bits heißt Carry-Flag. Diese 'Flagge' wird von der CPU unter anderem dann gesetzt, wenn bei einer Addition ein Übertrag entsteht. Übertrag heißt, daß das Ergebnis größer als 255 wurde und somit nicht mehr in ein 8-Bit-Register paßt. Sein Inhalt wird auch bei einer Addition immer noch zum Ergebnis dazugezählt. Wir hätten es darum vor der Addition löschen sollen. Dazu dient der Befehl

CLC

(CLear Carry flag), den wir also noch vor unser Programm schreiben müssen. Bislang besteht unser Programm also aus den Befehlen

```
CLC
LDA #6
ADC #3
```

Laß' gehn!

Wir wollen uns jetzt auf dem Bildschirm ansehen, wie es arbeitet. Gehen Sie dazu ins Hauptmenü der 'Assembler-Schule', drücken Sie F3 und wählen Programm 1 aus. Sie sehen (hoffentlich) ein Listing unseres ersten Programms. Drücken Sie nun F7. Am oberen Bildschirmrand erscheinen die Zeilen

```
A  X  Y  SP  PC    NV-BDIZC
00 00 00 fe  c000  00110000 clc
```

Ganz links, unter dem A, wird der Inhalt des Akkus (hexadezimal) angezeigt, unter den Buchstaben NV-BDIZC in binärer Form der des Status-Registers. Sein niederwertigstes Bit — unter dem C — ist die Carry-Flagge. Rechts steht der als nächstes auszuführende Befehl. Um ihn ausführen zu lassen, drücken Sie die Leertaste.

An den Register-Inhalten ändert sich dadurch nichts. Die Carry-Flagge war ja schon gelöscht, aber sicher ist sicher.

Nach dem nächsten Druck auf die Leertaste wird der Befehl LDA #6 ausgeführt. Der Akku enthält nun die Zahl 6. Noch mal SPACE - die Addition wird ausgeführt, der Akku enthält nun 9. Die Carry-Flagge zeigt nach wie vor 0 an, ein Übertrag ist ja nicht aufgetreten.

In den Speicher schreiben

Der nächste Befehl, den Sie lernen sollten, heißt

STA $C008

und bedeutet 'STore Accu', also 'speichere den Inhalt des Akkus ab'. Und zwar in die Speicherstelle, deren Adresse hinter dem Befehl steht. In unserem Beispiel also an die Adresse C008 hexadezimal.

Springen Sie mit F7 wieder in den Editor, schreiben Sie diesen Befehl hinter unser bisheriges Programm und lassen es erneut ausführen. In der zweiten inversen Zeile in der unteren Hälfte des Bildschirms sollten Sie hinter der Adresse $C008 beobachten können, wie der Inhalt des Akkus dorthin kopiert wird.

Sie könnten erneut in den Editor gehen und andere Zahlen hinter die Befehle LDA und ADC schreiben. Die Doppelkreuze sollten Sie aber stehenlassen. Probieren Sie doch mal aus, wie groß die beiden Argumente höchstens sein können, ohne daß nach der Addition die Carry-Flagge gesetzt ist. Wie sieht das Ergebnis aus, wenn ein Übertrag auftritt?

Noch'n Programm

Wenn Sie diese Fragen geklärt haben, laden Sie bitte das zweite Programm in den Editor. (Mit STOP ins Hauptmenü, F3 drücken und 2 wählen.) Dieses Programm soll Ihnen etwas über Programm-Dokumentation zeigen.

Alle Zeilen, die mit einem Semikolon beginnen, enthalten einen Kommentar, der einzig und allein der Lesbarkeit des Programms dienen soll. Sie treten nach der Übersetzung durch den Assembler im eigentlichen Maschinen-Programm nicht mehr in Erscheinung.

Überall, wo in der zweiten Spalte (mit der Überschrift 'labl') etwas steht, wird ein Label definiert. Labels sind symbolische Namen für bestimmte Speicheradressen. Man kann sie auch als Variable für den Assembler auffassen. Sie enthalten als Wert die Adresse, hinter der sie stehen. Überall, wo im Quell-Code beispielsweise das Argument 'SUM1' auftaucht, setzt der Assembler dafür $C014 ein. Der Sinn dieser symbolischen Adressen ist in dem Artikel „Assembler als Hochsprache" an anderer Stelle in diesem Heft ausführlich erklärt.

Doch nun zu dem, was das Programm

tut: Den Befehl CLC kennen Sie schon. Der nächste Befehl heißt

LDA SUM1

SUM1 steht für 'erster Summand'. Dieser Name ist aber völlig beliebig. Er ist wie gesagt nur ein symbolischer Name für die Adresse $C014. Vor diesem Argument steht kein Doppelkreuz. Es handelt sich also nicht um ein unmittelbares Argument. (Die Zahl $C014 würde im übrigen auch gar nicht in den Akku passen.) Diese Adressierungsart heißt 'absolut'. Das bedeutet, der Akku soll mit dem Inhalt *der absoluten Adresse $C014 geladen* werden.

An der Adresse $C014 steht

W $1234

W ist kein Maschinensprache-Befehl, sondern eine Anweisung für den Assembler. Sie veranlaßt ihn, bei der Übersetzung an diese Adresse die dahinterstehende Zwei-Byte-Zahl zu schreiben. Dabei schreibt er das niederwertige Byte an die angegebene und das höherwertige an die folgende Adresse. Eine Zwei-Byte-Zahl in dieser Form wird auch 'Wort' genannt - daher der Befehl W.

In unserem Beispiel werden also nach der Übersetzung die Speicherstelle $C014 den Inhalt $34 und die Speicherstelle $C015 den Wert $12 haben.

Nach der Ausführung des Befehls 'LDA SUM1' wird der Akku also in unserem Beispiel den Inhalt $34 haben.

Die folgenden Befehle sollten Ihnen keine Rätsel mehr aufgeben: ADC SUM2 addiert den Inhalt der Speicherstelle, die den symbolischen Namen 'SUM2' hat, zum Inhalt des Akkus. STA SUMM speichert den Akku-Inhalt an die Adresse mit dem Label 'SUMM'.

An dieser Adresse taucht eine andere Assembler-Anweisung auf: Der Befehl

S 2

bewirkt, daß der Assembler an dieser Adresse zwei Bytes Platz (Space) läßt. Er füllt diese Adressen mit Null. Diese Adressen können vom Programm als Variable benutzt werden. In unserem Fall nehmen Sie das Ergebnis einer Zwei-Byte-Addition auf.

Der letzte unklare Befehl in diesem Programm ist

BRK

Er bedeutet BReaK (Abbruch). Wenn der Prozessor auf diesen Befehl trifft, hört er mit der Bearbeitung des Programmes auf und springt an eine (hardwaremäßig festgelegte) Adresse. Beim C64 wird dann normalerweise der Bildschirm gelöscht, und in der ersten Bildschirmzeile erscheint die Meldung 'READY.' Innerhalb unseres Simulators wird diese Meldung in der untersten Bildschirmzeile ausgegeben, und Sie können wieder in den Editor oder zum Hauptmenü springen.

Zieh ab!

Im dritten Beispiel-Programm geht es um die Subtraktion. Hier erst mal die neuen Befehle:

SEC

ist die Abkürzung für 'SEt Carry flag' und tut genau das Gegenteil von CLC. Mit diesem Befehl kann - zum Beispiel vor einer Subtraktion - in die Carry-Flagge eine 1 geschrieben werden.

SBC #$82

subtrahiert das Argument vom Inhalt des Akkus. Ausgeschrieben heißt der Befehl 'SuBtract from accu with Carry'. Dabei muß die Carry-Flagge gesetzt sein, sonst wird 'einer mehr' abgezogen. Tritt ein Unterlauf auf, so wird die Carry dabei gelöscht.

Im Prinzip funktioniert eine Subtraktion in Assembler genau wie eine Addition. Man muß nur die Befehle ADC gegen SBC und CLC gegen SEC austauschen.

Das Beispiel-Programm Nummer drei zeigt, wie ein Ein-Byte-Wert von einer Zwei-Byte-Zahl abgezogen wird. Dabei treten in einem Programm verschiedene Adressierungsarten auf — das ist völlig normal, die beiden anderen Programme waren etwas Besonderes.

Der Befehl SBC #0 sieht auf den ersten Blick einigermaßen überflüssig aus, ist er aber nicht. Wenn nämlich in der vorhergehenden Subtraktion vom Low-Byte ein Unterlauf aufgetreten ist, ist die Carry-Flagge an dieser Stelle gelöscht, und es wird nicht Null, sondern Eins vom High-Byte abgezogen. Und genau das ist ja beabsichtigt. Mit diesem 'Trick' sind Ein- und Zwei-Byte-Werte verknüpfbar. Diese Vorgehensweise entspricht dem spaltenweisen Subtrahieren von Dezimal-Zahlen, in der ein Unterlauf ja auch nach links übertragen wird.

Auch bei diesem Programm kann man übrigens die jeweils entgegengesetzten Befehle einsetzen, und es wird addiert. Wenn nämlich bei der Addition auf das niederwertige Byte ein Überlauf entsteht, wird die Carry gesetzt, und der Befehl ADC #0 addiert in Wirklichkeit Eins.

In der nächsten Folge der 'Assembler-Schule' werden Sie unter anderem lernen, wie negative Zahlen in Assembler behandelt werden. HS

Literatur

Christian Persson,
6502/65C02 Maschinensprache,
Verlag Heinz Heise GmbH,
Hannover 1983

Rodnay Zaks,
Programmierung des 6502,
Sybex-Verlag GmbH,
Düsseldorf 1981

Tips zur Assemblerprogrammierung

Assembler als Hochsprache

Teil 1: Symbole statt Zahlensalat

Eines kann wohl vorausgesetzt werden: zum Programmieren in Maschinensprache gehört ein Assembler, das „Einhacken" von Programmen im Monitor ist absolut tabu. Es gibt für den C64 eine fast unüberschaubare Anzahl von Assemblern auf dem Markt, zu den schnellsten gehört der INPUT-ASS (Ausgabe 6/86 — bißchen Eigenwerbung muß sein . . .). Die Beschreibung spezieller Assembler-Befehle bezieht sich auf den INPUT-ASS, andere Assembler werden sozusagen en passant erwähnt.

Es gibt Programme, die beginnen nach der ORG-Anweisung (dem Setzen des Programmzählers) sofort mit dem ersten Befehl. Allein daraus kann man schließen: dieses Programm taugt nichts! Es geht dann nämlich meist folgendermaßen los:

```
ORG $C000

LDA #00
STA $D021
```

Wer sich etwas im Innenleben des C64 auskennt, weiß, daß diese Befehlsfolge die Bildschirmfarbe auf schwarz setzt. Dies muß häufig mehrmals im Programm passieren, weil vielleicht ein bestimmtes Untermenü mit anderen Farben netter aussieht und anschließend auf die „Hauptfarbe" zurückgeschaltet wird. Und so steht dann LDA #00 und so weiter ziemlich oft im Programm. (Sämtliche Assembler-Befehle sind hier aus Gründen der besseren Lesbarkeit in Großschrift geschrieben, achten Sie beim Programmieren darauf, daß INPUT-ASS Kleinschrift verlangt!) Wenn jetzt jemals die Bildschirmfarbe von schwarz auf rot geändert werden soll, muß das ganze Programm durchforstet und der Befehl LDA #00 in LDA #02 geändert werden. Wäre unser Beispielprogramm ein gutes Programm, hätte vor dem ersten Befehl mindestens eine Zuweisung gestanden, nämlich:

Das Programmieren in Maschinensprache läßt dem Programmierer freie Wahl in Struktur- und Stilfragen. Man darf (beinahe) alles. Gerade deswegen ist Disziplin und Überlegung angesagt. Parallel zum Kursus „Maschinensprache" starten wir eine Artikelfolge, die Anfängern und Fortgeschrittenen Hinweise zur Erstellung überschaubarer und wartbarer Assemblerprogramme geben soll. BASIC-Programmierer sollten nicht gleich weiterblättern: einige prinzipielle Überlegungen lassen sich auf alle Programmiersprachen übertragen.

```
:HFARBE = 00 ; schwarz
```

Der Doppelpunkt als Labelanfang ist INPUT-ASS-spezifisch, statt des Gleichheitszeichens verlangen manche Assembler den Pseudo-Befehl EQU (kommt von Englisch EQUATE, gleichsetzen). Im Prinzip läuft's aber immer auf dasselbe hinaus: überall, wo im Programm der Bildschirm schwarz sein soll, wird kodiert:

```
LDA #HFARBE
STA $D021
```

Wenn jetzt die Hintergrundfarbe geändert werden soll, muß nur noch zu Anfang des Programmtextes die Zuweisung für HFARBE geändert werden. Hinzu kommt, daß das Programm dadurch auch lesbarer wird, weil man von dem Kürzel HFARBE eben relativ schnell auf die Bedeutung „Hintergrundfarbe" schließen kann. Aus Gründen der Lesbarkeit ist es auch sinnvoll, der Adresse $D021 einen Namen zu geben. So könnte der Anfang des Beispielprogramms folgendermaßen aussehen:

```
ORG $C000

:HFARBE = 00 ;schwarz
:BGRUND = $D021 ;Bildschirmhintergrund

LDA #HFARBE
STA BGRUND
```

Zahlen sind tabu

Übertragen auf Hochsprachen wie BASIC entspricht dies in etwa dem Definieren von Variablen. Daß sogar die Adresse $D021 einen Namen bekommt, mag pedantisch erscheinen, ist aber einigermaßen „selbstdoku-

```
org $c000
lda #$12
ldy #$c0
jsr $ab1e
ldx #16
lda #'*
:11 sta $072b,x
dex
bne 11
rts
b 147
b "  sterne,"
b "  sterne ..."
b 0
```

Ein Beispiel aus der Horror-Ecke der Assembler-Programmierung: nur absolute Zahlen, nicht kommentiert, Bezüge ins Programm zu Fuß berechnet.

mentierend". Apropos Dokumentation: Auf Kommentare sollte auch dann nicht verzichtet werden, wenn zum Zeitpunkt des Programmierens die Routinen völlig überschaubar erscheinen. Das stellt sich vier Wochen später meist völlig anders dar!

Aber es sollte ja um Symbole statt Zahlen gehen, also noch ein Beispiel. Es soll eine bestimmte Stelle auf dem Bildschirm direkt beschrieben werden. Eine ebenso häufige wie unkluge Kodierung ist:

```
LDA #$41
STA $0428
```

In die erste Spalte der zweiten Reihe des Video-RAM wird der Code für A geschrieben. Bekanntlich ist die Lage des Video-RAM im 64er verschiebbar, obendrein kann es sein, daß sich irgendwann im Verlauf der Programmentwicklung herausstellt, daß dieses Zeichen (oder eine ganze Reihe von Zeichen) besser in der dritten Reihe steht. Die meisten Assembler können arithmetische Ausdrücke berechnen, und das ist zuverlässiger und weniger arbeitsaufwendig, als dies selbst zu tun. Eine vernünftige Kodierung lautet darum:

```
:VIDEO = $0400 ;adresse bildschirmanfang
:ZLEN = 40 ;laenge einer zeile
:ZEILE = 2 ;ausgabe-zeile

LDA #'A
STA ZEILE★ ZLEN+VIDEO
```

Jetzt kann der Bildschirm beliebig verschoben werden, dies muß dem Assembler nur einmal durch eine veränderte Zuweisung für VIDEO mitgeteilt werden. Auch die Ausgabezeile kann schnell variert werden — die entsprechende Lage im Video-RAM berechnet der Assembler.

Rechnen muß der Rechner

Bei der Formulierung der Ausdrücke sollte man vorher einen Blick in die Anleitung werfen. Viele Assembler — so auch INPUT-ASS — werten Ausdrücke strikt von links nach rechts aus, nicht nach der üblichen Priorität der Operatoren, beispielsweise nicht nach der Regel „Punktrechnung geht vor Strichrechnung".

Auch Berechnungen, die sich auf Adressen innerhalb des Programms beziehen, sind natürlich Sache der Maschine. Ein weiteres Beispiel aus der Redaktionsarbeit, die nun mal zum großen Teil aus der Beurteilung eingeschickter Programme besteht:

```
#$00
:LOOP LDA $3000,X
JSR $FFD2
INX
CPX #17
BNE LOOP
RTS
```

Ein 17 Zeichen langer Text, der ab Adresse $3000 im Speicher vermutet wird, soll Zeichen für Zeichen ausgegeben werden. Die Routine mit dem Einsprung bei $FFD2 ist ein im C64-ROM installiertes Unterprogramm, das eben dieses tut. Das sollte auch durch einen entsprechenden Namen kenntlich gemacht werden. Schlimmer ist aber, daß durch diese Form der Kodierung der auszugebende Text nicht verändert werden kann, weder die Lage des Textes noch die Länge. Vernünftig ist:

```
:BSOUT = $FFD2

LDX #$00
:LOOP LDA TEXTANF,X
JSR BSOUT
INX
CPX #TEXTEND-TEXTANF
BNE LOOP
RTS
. . .
:TEXTANF B "Dies ist ein Text"
:TEXTEND          =
★
```

Die Bezeichnung BSOUT für diese Routine zur Zeichenausgabe stammt von Commodore. Wichtiger ist, daß jetzt der auszugebende Text in Lage und Länge variieren kann; er muß nur zwischen den Labels TEXTANF und TEXTEND stehen. (Und darf nicht länger als 255 Zeichen sein; das liegt aber an der Routine, nicht an der Art der Programmierung.) Die drei Punkte stehen für beliebigen Programmtext. Es wäre eigentlich noch geschickter, dem Text eine Endemarkierung zu verpassen, gängig ist ein Null-Byte. Das hat aber nichts mit dem Thema „den Assembler rechnen lassen" zu tun und soll deswegen nicht weiter ausgeführt werden.

Direkt in den Speicher

Weil es im letzten Beispiel vorkommt, zum Schluß noch ein paar Worte zu

den sogenannten Pseudo-Ops. Damit sind Befehle gemeint, die Funktionen des Assemblers steuern, im Gegensatz zu den Maschinensprahe-Befehlen, den „Mnemonics". Die Zuweisung an das Label TEXTEND durch = * bedeutet, daß TEXTEND den momentanen Wert des Programmzählers erhält, in diesem Fall den der ersten Adresse nach dem Text. Bei einigen Assemblern ist der Programmzähler über das $-Zeichen statt des * erreichbar.

Das B weist den Assembler an, ein oder mehrere Bytes abzulegen. (B ist die Kurzform für BYTE, manche Assembler wollen auch die Langform „sehen" oder BY, BYT, DEFB.) Das B mit dem in Anführungszeichen eingeschlossenen Text dahinter ist eine Besonderheit dieser Anweisung; jedes einzelne Zeichen wird als ein Byte interpretiert, bei anderen Assemblern steht in diesem Fall auch das Kürzel ASC (wie ASCII) als Anweisung. Häufig ist die Form

B $00

um ein Byte — hier mit dem Inhalt Null — beziehungsweise

B $41,$42,$43

um mehrere Bytes abzulegen. Statt absoluter Zahlen (natürlich nicht größer als 255!) können auch Symbole verwendet werden. Dies kommt allerdings häufiger bei der WORD-Anweisung vor. (Beim wenig geschwätzigen INPUT-ASS durch W abgekürzt.) Hier wird ein Wert in der Reihenfolge Low-Byte/High-Byte im Speicher abgelegt, die häufigste Verwendung ist das Anlegen von Adreß-Tabellen. Beispiel:

ORG $1000

:TEXT1 B "123",0
:TEXT2 B "345",0
. . .
:TEXTn

```
 org $c000
;starprint
;n zeichen in die x-te bildschirm
;-zeile mittig setzen. limits:
;0 < szeil < 25, 0 <= anzahl < 40
;berechnungen: der assembler!

;system-routinen
:cprint  =$a1e      ;print routine

;definitionen
:video   =$0400     ;bildschirm-adr.
:zeichen ='*        ;dieses zeichen
:anzahl  =16        ;so oft
:szeil   =20        ;in die zeile

       lda #<text   ;hol low-adr.
       ldy #>text   ;hol high-adr.
       jsr cprint   ;ausgeben

       ldx #anzahl  ;als zaehler
       lda #zeichen ;zum schreiben
:stelle=39-anzahl/2
:anfang=40*szeil+video+stelle
;mittig stellen!
:sloop sta anfang,x ;in screen
       dex          ;einen weniger
       bne sloop    ;schon alle?

       rts

:text  b 147        ;clr home
       b "  sterne, sterne ..."
       b 0          ;null=texteende
```

Symbolisch, klassisch, gut: alle Werte im Deklarationsteil zugewiesen, dadurch leicht zu dzurchschauen und zu ändern.

:TEXTTAB W TEXT1,TEXT2 W . . .,TEXTn

Über TEXTTAB könnte man nun der Reihe nach auf die Adressen der einzelnen Texte zugreifen, dort steht nämlich im assemblierten Code:

$00,$10,$04,$10, . . .

Wenn man die Textadressen lieber in zwei Tabellen ablegen möchte, eine Tabelle mit den Low-Bytes, eine mit den High-Bytes der Adressen, gibt es auch hierfür komfortable Assembleranweisungen: < und >. Das >-Zeichen steht für „High-Byte bilden", das <-Zeichen für „Low-Byte bilden". Am Beispiel der Textadressen sieht das folgendermaßen aus:

```
ORG $1000

:TEXT1 B "123",0
:TEXT2 B "345",0
...
:TEXTn

:TEXTTABL B <TEXT1,<TEXT2
B...,<TEXTn
:TEXTTABH B >TEXT1,>TEXT2
B...,>TEXTn
```

Im realen Code entspricht dies für TEXTTABL

$00,$04,...

für TEXTTABH

$10,$10,....

Eine Spezialität des INPUT-ASS ist die STORAGE-Anweisung (abgekürzt durch S), ein Befehl zum Freihalten von Bytes.

```
S 40
```

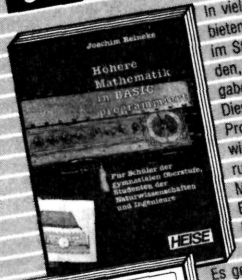
schreibt also nicht den Dezimalwert 40 in den Speicher, sondern hält 40 Bytes frei, die beispielsweise später als Puffer für die Ablage von Daten dienen können. Manche Assembler erlauben einen ähnlichen Effekt durch die Zuweisung *=*+40.

Wahrscheinlich entwickeln Sie mit einiger Übung noch weitere Tricks. Am besten, Sie nehmen sich dazu Ihren Assembler nebst Anleitung zur Hand und experimentieren ein bißchen. Wenn Sie die im Text aufgeführten Beispiele verwenden wollen, denken Sie daran, jede Routine mit RTS oder BRK zu begrenzen.

In den nächsten Folgen dieser Artikelreihe, die in lockerer Folge parallel zum Maschinensprache-Kurs erscheint, geht es um die Themen „Umgang mit Makros", „Bedingte Assemblierung" und „Strukturiertes Programmieren". Wenn man dies alles beherrscht, ist Assembler (fast) eine Hochsprache.

JS

HEISE/LUTHER
Bissendorfer Straße 8
3000 Hannover 61



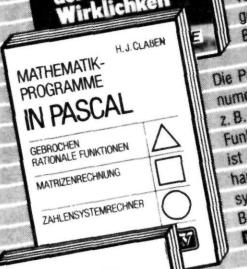
In vielen mathematischen Gebieten, die in der Schule und im Studium behandelt werden, kehren bestimmte Aufgabentypen stets wieder. Dieses Buch liefert BASIC-Programme dafür. Ferner wird stets eine kurze Einführung in die zugrundeliegende Mathematik gegeben.
Buch mit Diskette für C64
Best. Nr. 0527-8 **DM 98,00**

Es geht um Anwendungen aus der Mathematik, Physik und Chemie. Die ausführliche Darstellung der mathematischen Grundlagen wird durch Zahlenbeispiele so demonstriert, daß auch der Praktiker, der ohne Mathematik auskommen möchte, einen guten Zugang findet.
Best. Nr. 7008-9
DM 58,00

Dem mathematisch nicht speziell ausgebildeten Praktiker soll ein Zugang zu den gebräuchlichsten statistischen Verfahren eröffnet werden. Das Buch stellt 30 Programme zur Verfügung. Jedes ist ausführlich dokumentiert und mit Hinweisen zur Bedienung und Interpretation der Ergebnisse versehen.
Best. Nr. 0128-6
DM 39,80

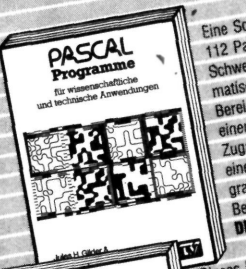
Dieses Buch bietet eine ausführliche und verständliche Einführung in die Grundlagen der Numerischen Mathematik und ihre Anwendung bei der BASIC-Programmierung. Alle Programme sind zusätzlich auch in einer Pascal- und FORTRAN-Version angegeben.
Best. Nr. 07042-9
DM 35,00

Dieses Buch untersucht Computer-Simulations-Modelle aus vielen Bereichen. Anhand zahlreicher Abbildungen werden die Ergebnisse der Simulationsverläufe kritisch diskutiert und die Grenzen der verwendeten Modelle herausgearbeitet.
Best. Nr. 0524-3 **DM 29,80**

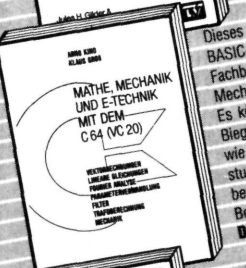


Die Programme behandeln numerische Probleme, wie z.B. das Ableiten beliebiger Funktionen. Darüber hinaus ist auch ein „Rechner" vorhanden, der beliebige Zahlensysteme verarbeiten kann.
Best. Nr. 0114-6
DM 38,80

Dieses Softwarepaket erlaubt unter Verwendung der hochauflösenden Graphik des C64 die anschauliche Präsentation der Resultate statistischer Analysen. Der Einsatz ist in nahezu allen Bereichen möglich.
Buch und Diskette für C64
Best. Nr. 13129-p
DM 89,00



Eine Softwarebibliothek mit 112 Pascalprogrammen. Der Schwerpunkt liegt im mathematischen und elektronischen Bereich. Anfänger finden hier einen sehr praxisbezogenen Zugang zu Pascal. Profis eine sofort einsetzbare Programmbibliothek.
Best. Nr. 9102-8
DM 49,80



Dieses Buch enthält acht BASIC-Programme aus den Fachbereichen Mathematik, Mechanik und Elektrotechnik. Es können z.B. Trafo- und Biegeträgerberechnungen sowie Torsions- und Biegebelastung bei Motor und Getriebewellen berechnet werden.
Best. Nr. 0115-4
DM 19,80

Dieses Buch bietet eine Einführung in die Matrizenrechnung. Klare Rechenvorschriften für Matrizennormierung, Determinantenberechnung und Matrizendivision sind auch für den Anwender sehr sinnvoll, der nicht in Pascal programmiert.
Best. Nr. 9156-7
DM 34,80

HEISE-Bücher erhalten Sie bei Ihrem Computer-, Elektronik- oder Buchhändler. Sollten unsere Bücher und Softwarepakete nicht im Fachhandel erhältlich sein, bitte über Kontaktkarte direkt anfordern und Verrechnungsscheck zzgl. DM 3,50 Versandkostenpauschale beifügen. Das ***Heise***-Gesamtprogramm kommt kostenlos mit.

Trainieren statt Pauken

Der Vokabeltrainer

Dieses Programm heißt nicht ohne Grund „Vokabeltrainer" statt beispielsweise „Wörterbuch" oder „Dictionary". Seine Stärke liegt nämlich in seinen ausgefeilten Lernoptionen. Die Fähigkeit, mehr als 2000 Vokabeln im Speicher zu halten, ist eher ein — natürlich erfreulicher — Nebeneffekt. Dazu Näheres später, erst einmal zur Bedienung.

Das Hauptmenü gliedert sich in folgende Punkte:

1. Vokabeln eingeben
2. Vokabeln verbessern
3. Vokabeln lernen
4. Vokabeln suchen
5. Vokabeln abspeichern & laden
6. Lernerfolg
7. Directory
8. Sonstiges

Auf über 2000 Vokabeln kann dieser vollständig in Maschinensprache programmierte Vokabeltrainer in Sekundenschnelle zugreifen. Seine eigentlichen Stärken aber zeigt dieses Programm als Partner beim Vokabelnlernen.

Durch Drücken einer Zifferntaste von 1-8 gelangen Sie in das entsprechende Unterprogramm.

Vokabeln eingeben

Die Vokabeln werden einfach der Reihe nach eingegeben. Zuerst das Fremdwort, dann die drei deutschen Bedeutungen. Sollte es keine zweite oder dritte Bedeutung geben, muß nur RETURN gedrückt werden. In das Hauptmenü gelangt man wieder, wenn bei der Aufforderung, das Fremdwort einzugeben, nichts eingegeben wird, oder durch die Funktionstaste f7. Sollte der Speicher voll sein, springt das Programm ins Hauptmenü zurück.

Vor der Eingabe fragt das Programm mit "Eingabekontrolle (j/n)" ab, ob kontrolliert werden soll, ob ein Fremdwort bereits in der Datei vorhanden ist. Eingabekontrolle meint, daß nach der Eingabe jedes Fremdwortes geprüft wird, ob dieses bereits in der Datei vorhanden ist. Ist dies der Fall, wird gemeldet: "Wort bereits vorhanden ! Weiter (j,n,t)?". Es sind jetzt drei Reaktionen möglich:

Taste j Das Programm fährt mit der Eingabe wie gewohnt fort. Das Fremdwort existiert dann zweimal in der Datei!

Taste n Das Programm erwartet die Eingabe eines neuen Fremdwortes.

Taste ↑ Die Vokabel kann im Verbesserungsmenü betrachtet oder verändert werden (Siehe unten!).

Trotz einer sehr schnellen Suchroutine dauert es bei vollem Speicher immer noch 6 Sekunden, bis dieser komplett durchsucht ist. Deshalb gibt es die Möglichkeit, selbst zu entscheiden, ob eine Eingabekontrolle gewünscht wird.

Vokabeln verbessern

Nach Drücken der Taste 2 befinden Sie sich im Verbesserungs-Modus. In der obersten Zeile sieht man die Nummer der aktuellen Vokabel. Darunter steht, wie gut diese Vokabel bereits gelernt wurde (dazu später mehr). Dann folgen das Fremdwort und die drei Bedeutungen. Der Cursor steht hinter dem Fremdwort, das nun verbessert werden kann. Nach Drükken der RETURN-Taste kann das nächste, darunterliegende Wort verändert werden. Mit SHIFT und RETURN wird die verbesserte Vokabel in den Speicher übernommen. Gelöscht werden kann die Vokabel mit C= und I. (Mit C= ist die sogenannte Commodore-Taste, neben der linken Shift-Taste, gemeint. Besitzer umgebauter Betriebssysteme — zum Beispiel „Turbo-Access" — erleben dabei gelegentlich den Effekt, daß das Programm anscheinend „hängt". Dies hängt damit zusammen, daß unter diesen Umbauten die Bildschirmausgabe mit der C=-Taste gestoppt werden kann. Abhilfe: ein nochmaliges kurzes Betätigen dieser Taste.)

Die Option „Vokabeln verbessern" kann auch dazu „mißbraucht" werden, sich alle Vokabeln einer Datei auf dem Bildschirm ausgeben zu lassen: Mit der Funktionstaste f1 blättert man um eine Vokabel weiter, mit f3 zurück, mit f2 geht's in 20er Schritten voran, mit f4 zurück.

Zusätzlich ist in diesem Unterpunkt die Druckeransteuerung eingebaut: Mit C= und f wird die aktuelle Vokabel formatiert, mit C= und d unformatiert ausgegeben. Formatiert heißt, daß das Fremdwort und die deutsche Bedeutung immer den gleichen Abstand haben. Mit C= und V beziehungsweise C= und C wird entsprechend fortlaufend ausgedruckt. Bei ausgeschaltetem Drucker kann diese Funktion auch als Suchlauf dienen.

Wird f5 gedrückt, durchsucht das Programm den Speicher nach dem momentan auf dem Bildschirm sichtbaren Fremdwort. Gesucht wird ab der gegenwärtigen Position, das heißt, wenn gerade die Vokabel Nr. 15 sichtbar ist, beginnt die Suche mit Vokabel Nr. 16. Auf diese Weise ist es sehr einfach möglich, Vokabeln zu suchen, die mehrfach vorhanden sind.

f7 führt wie immer zurück ins Hauptmenü. Als letztes eine kleine Gedächtnisstütze: Nach Drücken der Taste C= und i wird die eben beschriebene Tastaturbelegung auf dem Bildschirm angezeigt.

Vokabeln lernen

So, nun ist es an der Zeit, die Lernmethode zu erläutern: Die Grundidee war, den Computer in die Lage zu versetzen, Vokabeln gezielt abzufragen; also zu unterscheiden, ob eine Vokabel bereits gelernt wurde.
Jeder Vokabel ist als „Bekanntheitsmaßstab" eine Zahl von null bis drei zugeordnet. Zunächst gilt jede Vokabel als „nicht gelernt", und diese Zahl wird auf drei gesetzt. Jedesmal nun, wenn zu diesem Wort die richtige Bedeutung angegeben wurde, wird diese Zahl um eins erniedrigt, andernfalls wieder auf drei zurückgesetzt. Das Programm fragt jede Vokabel mindestens dreimal ab, dann gilt sie als „sehr gut gelernt". Da diese Information auch gleichzeitig mit den Vokabeln abgespeichert wird, kann man jederzeit bestimmte schwer erlernbare Wörter nachlernen. Der Platzbedarf für diese Zusatzinformation ist gering, weil dafür nur zwei Bits benutzt werden. Wenn man unter Zeitdruck steht und am nächsten Tag eine Prüfung schreiben muß, aber nichts gelernt hat, kann man so schnell die schweren (noch nicht gelernten) Wörter wiederholen. Im Prinzip ist dies nichts anderes als ein Herausfiltern der leicht erlernbaren Wörter. Es ist deshalb übrigens auch nicht sinvoll, die Vokabeln dann abzuspeichern, wenn sie alle sehr gut gelernt sind.

Wenn Sie nun durch Drücken der Taste 3 ins Vokabel-Lern-Menü gelangen, dürfte es kein Problem mehr sein, dies zu verstehen. Je nachdem, welche Taste Sie Drücken, werden die entsprechenden Vokabeln herausgesucht und abgefragt. Wurde zum Beispiel eine Taste von 2 bis 4 gedrückt, fragt der Computer so lange ab, bis es keine Vokabeln mehr gibt, die dem Abfragekriterium entsprechen (zum Beispiel noch nicht gelernt zu sein). Haben Sie hingegen die Taste 1 gedrückt, fragt das Programm so lange ab, bis Sie es mit der Taste f7 erlösen.

Zuvor können Sie noch entscheiden, ob nach dem Fremdwort oder der deutschen Bedeutung gefragt werden soll. Auf Erfolgs-Sounds oder -Graphiken wurde bewußt verzichtet, damit niemandem die Lust am Lernen vergeht. Eine Lernerfolgs-Auswertung gibt es deshalb erst gesondert im gleichnamigen Unterpunkt. Auf eine unzutreffende Antwort wird übrigens nicht mit der Meldung „falsch" reagiert, sondern nur mit der Bemerkung „so sorry . . .". Ihre Antwort kann ja durchaus richtig sein, aber das Programm kennt diese Bedeutung nicht. Achten Sie bitte auf Groß-/Kleinschreibung!

Vokabeln suchen

Hier kann entweder das Fremdwort oder die deutsche Bedeutung einge-

geben werden. Der Computer sucht dann den ersten Vokabelsatz, in dem das eingegebene Wort vorkommt. Anschließend erfolgt ein Sprung in das Verbesserungs-Menü, wo das Wort zur Bearbeitung oder Betrachtung zur Verfügung steht. Im Gegensatz zu anderen Untermenüs ist ein Abbrechen dieser Funktion nur mit der STOP-Taste möglich. (Das geht um etwa 15% schneller, als wenn man die f7-Taste abfragen würde). Mit f5 kann die Datei weiter nach dem eingegebenen Begriff durchsucht werden. Erscheint dann die Meldung „nicht gefunden", heißt dies natürlich, daß dieses Wort kein zweites (oder drittes) Mal in der Datei vorhanden ist.

Beim Durchsuchen der Vokabeln werden nur jeweils soviel Zeichen verglichen, wie das Suchwort enthält. Das heißt, über die Eingabe von „do" wird sowohl das Wort „do" selbst gefunden als auch „door", „double" und so weiter.

Vokabeln abspeichern & laden

Zunächst erscheint folgendes Untermenü:

1. abspeichern
2. laden
3. nachladen

Mit f7 gelangt man wieder in das Hauptmenü. Die Vokabeln werden als serielles Daten-File abgespeichert und geladen. Dies dauert zwar ein wenig länger, hat aber den Vorteil, daß man direkt aus dem RAM unter dem ROM abspeichern kann. Beim Nachladen werden die Vokabeln an die bereits im Speicher vorhandenen angehängt, „Laden" überlädt eine eventuell im Speicher vorhandene Datei! Sollte ein Fehler auftreten, wird das Speichern beziehungsweise Laden abgebrochen und eine entsprechende Fehlermeldung ausgegeben. Ein Abbrechen ist natürlich auch mit der STOP-Taste möglich.

Lernerfolg

Zunächst werden die wichtigsten Daten über die Vokabeln ausgegeben. Auf Tastendruck werden dann die „Lernerfolgsdaten" grafisch aufbereitet.

Directory

Diese Funktion ist nur mit einer Diskettenstation möglich. Das Auflisten der Directory kann mit der Leertaste angehalten werden.

Sonstiges

In diesem Untermenü kann auf zahlreiche Sonderfunktionen zugegriffen werden:

1. Massenspeicher Durch Drücken der Taste 1 kann man den Massenspeicher verändern. Es stehen drei Möglichkeiten zur Verfügung: Datasette, Diskette und SuperTape. Vor der Benutzung von SuperTape muß dieses vorher geladen und gestartet worden sein. (SuperTape ist ein extrem schnelles Lade- und Speicherprogramm für Kassette, das in INPUT64 4/85 veröffentlicht wurde.)

2. Hintergrundfarbe und 3. Zeichenfarbe Nach dem Drücken dieser Tasten wird der Farbwert um eins erhöht (aus Schwarz wird Weiß, dann Rot und so weiter).

4. Speicher löschen Alle Vokabeln im Speicher werden unwiderruflich gelöscht.

5. Vokabel-Reset Wenn Sie diese Taste gedrückt haben, wird der Vokabeltrainer im folgenden alle Vokabeln als nicht (beziehungsweise als nicht mehr) gelernt interpretieren.

6. Floppy-Befehl Hier kann man einen beliebigen Befehl zur Diskettenstation senden.

Die Befehle werden in der üblichen Syntax eingegeben, und zwar nur der Befehls-String selbst. Beispiel:
N:DATEIEN,77
formatiert eine Diskette.

7. .Rückkehr ins BASIC Wenn Sie wieder ins BASIC wollen, können Sie diese Taste benutzen. Das Programm läßt sich mit einem SYS 2118 ohne Datenverlust neu starten.
Auch dieses Untermenü können Sie mit f7 verlassen.

Programmtechnisches

Unter dem Vokabeltrainer ist der im 64er verfügbare Speicherplatz folgendermaßen aufgeteilt:

Bereich	Nutzung
$0000-$0800	Zeropage, Bildschirmspeicher, Kassettenpuffer usw.
$0801-$BFFF	Programm und Daten, Teil 1
$C000-$CFFF	Freier Platz, z.B. für SuperTape
$D000-$FFFF	Vokabelspeicher, Teil Nr.2

Die folgenden Speicherzellen werden in der Zeropage genutzt:

$F7/$F8	Hauptzeiger
$F9/$FA	Zwischenspeicher für den Hauptzeiger
$22/$23	Universalzeiger
$CC	Steuerung des Blinkens
$90	Status

Außerdem werden noch einige Zeiger im Zusammenhang mit der Eingabe von Wörtern und beim Abspeichern genutzt. Diese werden jedoch nur indirekt verändert (Cursor setzen, Farb-RAM berechnen und so weiter)

Strukturell ist das Programm wie das Hauptmenü aufgebaut. Grundidee des Vokabeltrainers war es, möglichst viel Platz für die Vokabeln zur Verfügung zu haben. Deswegen wird nicht für jedes Zeichen ein ganzes Byte „verschleudert", sondern nur sechs Bits. Dadurch ergibt sich folgende Zuordnung:

ASCII	Interncode	Art
65-90	11-36	Kleinbuchstaben
193-218	37-62	Großbuchstaben
frei	1-10	unbenutzt
———	0,63	programm-benutzt

„Teufelchen"

Beim Verbessern hat man gelegentlich den Eindruck, daß der Computer abgestürzt sei. In fast 100% der Fälle wird man aber versehentlich die Taste f5 gedrückt haben. Der „Fehler" läßt sich also jederzeit mit der RUN STOP-Taste beheben.

Sollte einmal eine sequentielle Datei geladen werden, die nicht vom Vokabeltrainer stammt, so merkt man dies daran, daß das Programm nicht mehr aufhört zu laden. Bitte drücken Sie in einem solchen Fall die STOP-Taste und löschen den Speicher (Sonstiges!). Fehler des Diskettenlaufwerkes werden dagegen sofort abgefangen und anschließend der Fehlerkanal ausgelesen.

Sollte das Programm von selbst in das Hauptmenü zurückkehren, so ist meistens ein voller Speicher die Ursache. Innerhalb von INPUT64 steht nur ein kleiner Teil des 64er-Speichers für die Ablage von Daten zur Verfügung. Um den ganzen Speicher ausnutzen zu können, muß der Vokabeltrainer auf eine eigene Diskette/Kassette abgespeichert werden (CTRL und s). Löschen Sie bitte vor dem Abspeichern den Vokabelspeicher („Sonstiges" wählen, dann 4 wie „Speicher löschen")!

Datei schon dabei

Zu einem Vokabeltrainer gehört natürlich auch eine Übungsdatei, Computerbesitzer brauchen (mindestens) deren zwei. Deswegen finden Sie im Magazin ein zweites Modul unter dem hochtrabenden Namen „Datenbank", das die Dateien „Englisch all." und „Englisch EDV" enthält. Die 100 eher willkürlich zusammengestellten Vokabeln, die Sie vorfinden, wenn Sie den Vokabeltrainer innerhalb von INPUT starten, dienen mehr ersten spielerischen Versuchen.

„Englisch all." enthält die 500 wichtigsten Vokabeln aus dem englischen Grundwortschatz; in „Englisch EDV" finden Sie 400 gängige Begriffe aus der Computerwelt. Das Studium englischsprachiger Handbücher dürfte damit kein Problem mehr sein. (Beide Dateien wurden übrigens von einer „Fachfrau" erstellt, Renate Ohmen, der Autorin des Kurses „Englische Grammtik".) Die Aufteilung in zwei Dateien hat sich aus mehreren Gründen als sinnvoll erwiesen:

Erstens haben Begriffe in technischen Fachsprachen oft eine völlig andere Bedeutung als in der Umgangssprache, man denke nur an das Wort „memory". Wenn Sie das im Rahmen eines Artikels einer Tageszeitung mit „Speicher" übersetzen, klingt das einigermaßen daneben; und vom „Gedächtnis" eines Computers zu reden ist allenfalls auf der allegorischen Ebene akzeptabel.

Zweitens sollten die Dateien nicht übermäßig groß werden, wenn man jemals eine positive Lernerfolgs-Auswertung sehen möchte. Das Programm kann zwar über 2000 Vokabeln verwalten, bei diesen Datenmengen brauchen Sie aber Jahre, um den Anteil der nicht gelernten Vokabeln auch nur unter 50 Prozent zu drükken.

Außerdem können Sie die beiden mitgelieferten Dateien als Muster für das Anlegen eines eigenen Vokabelschatzes betrachten:

Dateien sollten streng in einer Richtung angelegt werden, entweder englisch/deutsch oder deutsch/englisch. Im letzteren Fall muß erst das deutsche Wort als „Fremdwort" eingegeben werden, dann die englische(n) Bedeutung(en). (Daß für „englisch" auch eine andere Fremdsprache stehen kann, versteht sich wohl von selbst.)

Bei unregelmäßigen Verben sollten alle drei Zeiten eingegeben werden, da diese dann gleich mit gelernt werden können.

Und: Es ist nicht sinnvoll, einen Vokabeltrainer als Wörterbuch zu mißbrauchen. Zu einem Wörterbuch gehören auch Kommentare, Querverweise und Lautschrift — alles Dinge, die in einem Buch besser realisiert werden können oder mindestens die Gerätekonstellation Personalcomputer mit Festplatte voraussetzen.

Zum Schluß noch ein Trost für treue INPUT-Leser, die bereits umfangreiche Vokabeldateien für das in INPUT 2/85 veröffentlichte Programm „Dictionary" erstellt haben: Der Autor schreibt an einem Wandler, der Dictionary-Dateien in Vokabeltrainer-Dateien umsetzt. Wir werden dieses Programm natürlich so schnell wie möglich veröffentlichen. Oliver Kraus/JS

Schatzsuche

Nützliches aus dem Betriebssystem

Wer sich einen C64 kauft, ahnt meist nicht, daß er unter anderem 16 (Kilobyte) Software erwirbt, die sich auch für eigene Programme nutzen läßt. Hierin sind einige sehr praktische Programme verborgen, die sich auch von BASIC aus aufrufen lassen. In den 64er Tips 11/85 hatten wir bereits die Tür zu diesem Programmspeicher ein wenig geöffnet. Diesmal werden wir ein paar besonders interessante Winkel durchsuchen.

Da man sich meist im Betriebssystem nicht so leicht zurechtfindet, sorgen wir erst einmal für Überblick: Der C64 hat 64 Kilobyte RAM, die ihm wohl den Namen gegeben haben. Diese 64 Kilobyte sind 64*1024=65536 einzelne Bytes, die der Einfachheit halber von 0 bis 65535 durchnumeriert werden. Mit Hilfe dieser Adressen kann jede Position des Speichers gelesen und beschrieben werden. Damit ist der Bereich, auf den der Mikroprozessor 6510 zugreifen kann, eigentlich schon vollständig ausgenutzt, trotzdem stehen im C64 noch weitere 20 Kilobyte ROM zur Verfügung, die unter anderem das Betriebssystem enthalten. ROM-Speicher (Read Only Memory) können selbstverständlich nur gelesen werden. Damit der Rechner nach dem Einschalten auf die Programme in den ROM-Bausteinen zugreifen kann und überhaupt vernünftig läuft, sind einige Bereiche des RAM-Speichers vom ROM-Bereich überblendet, wodurch der versteckte RAM-Bereich nicht ohne weiteres erreichbar ist (siehe Bild 1).

Systeme

Was bisher allgemein die Bezeichnung „Betriebssystem" getragen hat, besteht, genauer gesagt, aus den Komponenten BASIC-Interpreter, Zeichensatz und KERNAL-Betriebssystem (oder kürzer: KERNAL). Der BASIC-Interpreter belegt 8 Kilobyte mit seinen Maschinen-Programmen, die die Aufgabe haben, ihrem Namen gerecht zu werden: Sie interpretieren Ihr BASIC-Programm oder Ihre Befehle im Direktmodus.

Das KERNAL besteht ebenfalls aus Maschinen-Programmen, die sich jedoch mehr um die Steuerung des Rechners und seiner Umgebung kümmern, wie Bedienung von Floppy, Datasette, Drucker, Bildschirm, Tastatur und anderen Geräten. Beim Laden und Speichern von Programmen, dem Bildschirmaufbau oder dem Cursor-Blinken erledigt das KERNAL den wesentlichen Teil der Arbeit.

Im Teamwork

Tippen Sie etwa den Befehl LOAD„PROGRAMM",8 ein, liest der BASIC-Interpreter einen Buchstaben nach dem anderen, bis er den Befehl LOAD erkannt hat. Anschließend übernimmt die entsprechende Maschinen-Routine den aktuellen Programmnamen (PROGRAMM) und die Gerätenummer (8). Der Interpreter ruft die zugehörige Routine des KERNAL auf, die den eigentlichen Ladevorgang ausführt. Ist das Programm geladen, übergibt das KERNAL die Kontrolle wieder an den Interpreter, welcher nun feststellt, ob ein Fehler aufgetreten ist. Wenn dem so ist, bekommen Sie eine der mehr oder weniger verständlichen Fehlermeldungen, sonst meldet sich der C64 mit dem bekannten „READY.".

Funktionelles

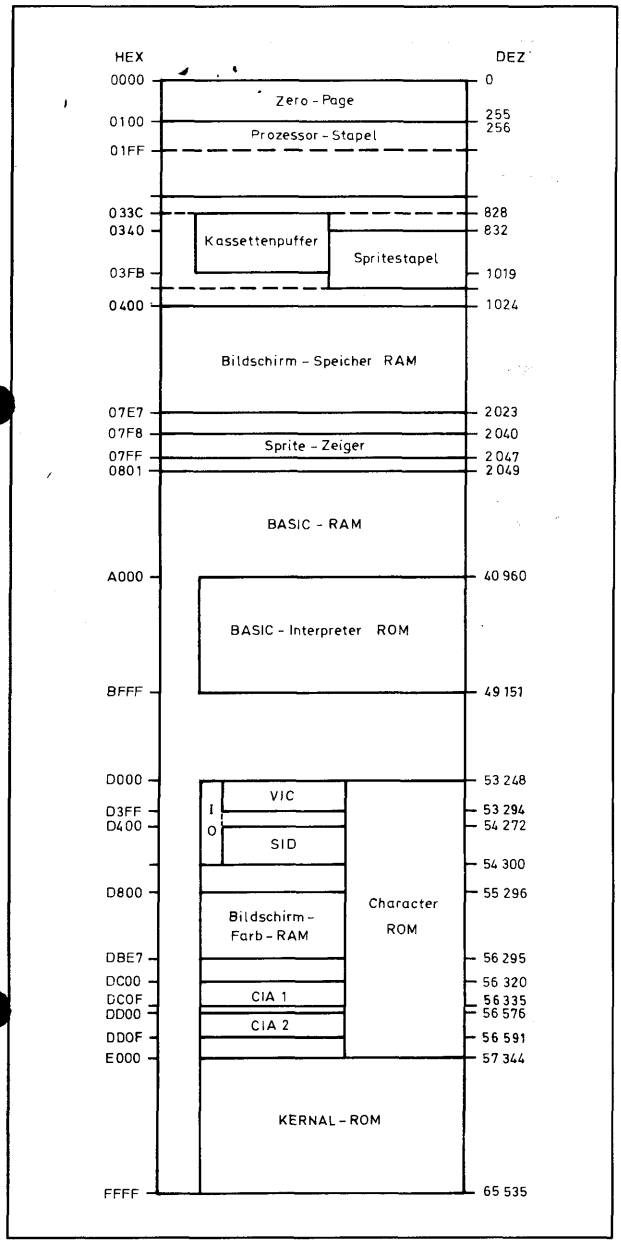
Dieser Dialog mit dem KERNAL läßt sich sinnvoll auch von der Benutzeroberfläche aus bewerkstelligen. Man umgeht dabei eine ganze Reihe von Kontroll- und Prüf-Funktionen des Interpreters, muß jedoch selbst dafür sorgen, daß die Mitteilungen an das KERNAL richtig sind. Das sonst recht karge BASIC V2 des C64 stellt gleich zwei Befehle zum Aufruf von Maschinen-Programmen zur Verfügung:

SYS XXXXX
USER (X)

Wir beschränken uns hier auf den SYS-Befehl. Der USER-Befehl wurde in INPUT 64 5/86 unter der Rubrik 64er Tips genauer erklärt. Mit dem SYS-Aufruf wird ein Maschinen-Programm an der Adresse XXXXX gestartet, was sicher den meisten Lesern vertraut ist. Weniger bekannt hingegen ist die Funktion der Speicherstellen 780 bis 783, welche die Register des Mikroprozessors darstellen. Dabei entsprechen:

780	Akkumulator
781	X Register
782	Y Register
783	Prozessorstatus Register

Bevor das Maschinen-Programm endgültig starten darf, nehmen die Prozessor-Register die hier abgelegten Werte an, bei der Rückkehr werden die Registerwerte hier gespeichert.



Speicheraufbau im Überblick

Als Anwender brauchen Sie sich nur zu merken, daß über diese Speicherplätze gewöhnlich der Datenaustausch mit den aufgerufenen Routinen stattfindet. Je nach Einsprungspunkt können aber auch andere Speicherstellen für die Übergabe verantwortlich sein. Damit Sie leichter finden, was Sie brauchen, sind alle im folgenden beschriebenen Einsprünge als eine Rezeptsammlung einer Tabelle zusammengefaßt. Dort sind auch die Routinen aufgeführt, die wir Ihnen schon in Ausgabe 11/85 ausführlich vorgestellt haben.

Da die meisten Routinen einen 16-Bit-Wert getrennt nach Low- und Highbyte verlangen, muß zuerst eine kompetente Lösung für dieses Problem her. Es bietet sich hier das Definieren von zwei Funktionen an, die bei Übergabe einer Zahl deren Low- bzw. Highbyte berechnet. Am Anfang Ihres BASIC-Programms sollten die Zeilen aus Listing 1 stehen. Mit FN L(A) erhält man dann im Programm das Lowbyte von A, mit FN H(A) das Highbyte.

Absolut

Gerade bei der Beschäftigung mit hochauflösender Grafik, aber auch beim Umgang mit Maschinen-Programmen, sind die Funktionen des Ladens an eine bestimmte Adresse oder Speicherns eines bestimmten Bereichs unverzichtbar, die als BASIC-Befehle leider nicht zur Verfügung stehen. Nun teilt der Interpreter dem KERNAL mit, an welche Adresse geladen werden soll, und das gilt es zu umgehen. Zuerst benötigen wir die KERNAL-Routine, die File-Name und Gerätenummer setzt,wie sie in Listing 2 verwendet wird.

Sie haben richtig gelesen: Zwischen der Adresse und dem String darf kein Komma stehen. Das hat zur Folge, daß Adresse und String nicht beide in einer Variable stehen dürfen, weil dies

zu einem TYPE MISMATCH ERROR führt. Wollen Sie ein Programm unter „Name" von der Diskette an die Adresse A laden, können Sie dies mit Listing 2 erreichen. Statt LOAD wird VERIFY durchgeführt, wenn Sie in Zeile 140 die Null durch eine Eins ersetzen. Da hier der Interpreter nicht benutzt wird, kann das laufende Programm auch nicht mit einer Fehlermeldung bei Mißerfolg abgebrochen werden, etwa wenn die Floppy nicht angeschlossen worden ist.

Beim Speichern eines Bereiches müssen ebenso File-Name und Gerät angegeben werden, wie es in Listing 3 dargestellt ist. Hierbei müssen Anfangsadresse A und Endadresse E angegeben werden. Diese Funktionen arbeiten auch mit Kassette im Commodore-Format. Bei SuperTape werden jedoch andere Routinen benutzt, so daß dieses Verfahren dort nicht funktioniert. Genaueres steht in den Tips 2/87.

Eine weitere interessante Anwendung dieser Technik ist das Abspeichern des Variablenspeichers eines Programms, um so seinen momentanen Zustand zu sichern.

Ein wahres Füllhorn

Der sonst so unscheinbare DIM-Befehl bietet einen besonderen Leckerbissen: er enthält eine Routine, die einen beliebigen Speicherbereich mit einem beliebigen Byte füllen kann. Normalerweise wird damit das dimensionierte Variablenfeld gelöscht, aber der phantasiereiche Anwender findet bestimmt weitere Einsatzmöglichkeiten. Am naheliegendsten ist sicher das von BASIC aus problematische Löschen einer HiResgrafik.

Da die Routine noch anderweitig verwendet wird, ist der Einsprungspunkt und damit die Parameterübergabe etwas undurchsichtig, was aber die Leistungsfähigkeit nicht beeinträchtigt, wie Sie in Listing 4 feststellen können. Dieser Programmteil füllt den Bereich von A bis E mit der Zahl XXXX. POKE12,1 gaukelt der Routine vor, sie sei vom DIM-Befehl aufgerufen worden. Springt man bereis bei 45760 ein, wird der Poke an die Adresse 780 überflüssig, als Füllbyte wird dann Null genommen.

Einen Nebeneffekt hat die Sache aber doch noch. Die Routine möchte unbedingt noch mitteilen, wie viele Bytes gerade gelöscht wurden. In den Speicherstellen 95/96 steht die Adresse der Speicherstelle minus zwei, wo diese Anzahl abgelegt wird. Da in 95/96 gewöhnlich 0 steht, werden die Speicherstellen 2 und 3 verwendet. Erfahrenen Programmierern läutet jetzt eine Alarmglocke im Ohr! Ein Blick ins C64-Handbuch auf Seite 160 bestätigt den Verdacht: Die Speicherstellen 3 und 4 beinhalten den „Vektor zur Umrechnung Float-Fixed". Der darf doch sicher nicht zerstört werden? Die Routine ist also unbrauchbar? Nein, verblüffenderweise werden die so gewichtig anmutenden Vektoren im gesamten Interpreter nicht verwendet! Hier scheint es sich um ein Überbleibsel aus vergangenen Zeiten zu handeln. Die Probe aufs Exempel bestätigt es: Wir haben kein BASIC-

Tabelle Listings

Listing 1:

```
10 DEF FN H(X)=INT(X/256)
20 DEF FN L(X)=X-FN H(X)*256
```

Listing 2:

```
100 A=3*4096
110 SYS57812 "Name",8
120 POKE781,FNL(A)
130 POKE782,FNH(A)
140 POKE780,0
150 SYS 65493
```

Listing 3:

```
100 A=3*4096
110 E=5*4096
120 SYS57812 "Name",8
130 POKE193,FNL(A)
140 POKE194,FNH(A)
150 POKE174,FNL(E)
160 POKE175,FNH(E)
170 SYS62954
```

Listing 4:

```
100 POKE88,FNL(E-FNL(E-A))
110 POKE89,FNH(E-FNL(E-A))
120 POKE113,FNL(E-A)
130 POKE114,FNH(E-A)
140 POKE780,Byte
150 POKE12,1
160 SYS45762
```

Listing 5:

```
POKE2050,1
SYS42291
POKE46,PEEK(35)-(PEEK(781)>253)
POKE45,PEEK(781)+2 AND 255
CLR
(Eingabe im Direktmodus)
```

Programme nach Bedarf

Tabelle der KERNAL-Routinen

```
SYS 62913   gibt Filenamen aus
SYS 58778   initialisiert VIC
SYS 64931   initialisiert CIA
SYS 65418   initialisiert I/O-
            Vektoren

POKE781,Zeile    Bildschirmzeile
SYS59903         löschen

POKE781,Zeile    Bildschirmzeile
POKE782,Spalte   bis Spalte
SYS59905         löschen

SYS59626         Bildschirm-
                 scrollen

SYS58648         Bildschirm-Reset
```

```
poke783,0        Cursor setzen
POKE781,Zeile
POKE782,Spalte
SYS 65520

poke783,1        Cursorposition
SYS 65520        holen
Zeile=PEEK(781)
Spalte=PEEK(782)

SYS58592         Auf Tastendruck
                 SPACE oder C=
                 warten

100 POKE95,0     BASIC-Interpreter
110 POKE96,160   ins RAM kopieren
120 POKE90,0
130 POKE91,192
140 POKE88,0
150 POKE89,192
160 SYS41919
```

Programm gefunden, das nicht lief, wenn diese Vektoren auf Null gesetzt wurden.

Keine Panik

Wenn es wirklich passiert ist, daß Sie aus Versehen NEW getippt oder den Reset-Taster betätigt haben, gilt zuerst: Ruhe bewahren. Geben Sie die Befehle aus Listing 5 ein. Die Befehle ohne Zeilennummer natürlich im Direkt-Modus. Danach ist jedes normale BASIC-Programm wieder lauffähig. Das Ganze arbeitet mit der Routine, die nach dem Laden eines Programms die Verknüpfungs-Zeiger der BASIC-Zeilen neu berechnet und das Programm von vorn bis zum Schluß durchgeht. Dabei werden bereits die ersten beiden verlorenen Bytes des Programms restauriert. Die nachfolgenden beiden Zeilen setzen den Zeiger auf das Ende des BASIC-Programms neu, und es kann wieder gestartet werden.

Maßnahmen

Schon in Ausgabe 11/85 haben wir Ihnen eine Routine zum Löschen einer Bildschirmzeile vorgestellt. Sie ist in der Tabelle noch einmal aufgeführt. Zwei Byte später eingesprungen, eröffnet sich die Möglichkeit, auch die Spalte zu bestimmen, bis zu der die Zeile gelöscht wird. SYS 62913 gibt den zuletzt benutzten File-Namen aus, was nützlich sein kann, wenn man auf Fehlersuche ist. SYS 58778 initialisiert den Videochip, SYS 64931 initialisiert die CIA (Interrupt u.a.), und SYS 65418 biegt die I/O-Vektoren zurecht.

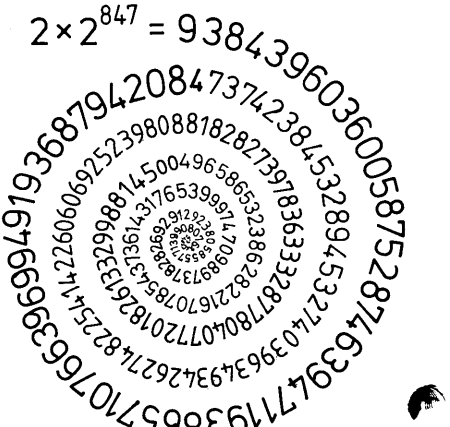
Diese Einsprünge lassen sich verwenden, wenn schnell geordnete Verhältnisse gebraucht werden. Schließlich enthält der Interpreter noch eine Blockverschiebe-Routine, die aber eine universelle Verwendung nicht zuläßt. Eine sinnvolle Anwendung gibt es, wenn der Interpreter ins RAM kopiert werden soll. Statt der 30 Sekunden, die die übliche Schleife benötigt, geht es mit den Zeilen in Listing 6 in einer Sekunde.

Gebührenfreie Verwarnung

Nachdem Sie so viel über die Vorzüge der Betriebssystem-Routinen erfahren haben, ist es Zeit für ein mahnendes Wort: Übertreiben Sie es nicht mit direkten Einsprüngen. Sie machen das Programm damit unübersichtlicher für andere und die Fehlersuche aufwendiger. Außerdem gelten die hier genannten Einsprungadressen natürlich nur für den C64 im Originalzustand. Haben Sie ein geändertes Betriebssystem, funktioniert möglicherweise nicht mehr alles. Hier geht probieren über studieren, durch Eingaben an der Tastatur können Sie den Rechner nicht beschädigen. Auch wenn er abstürzt und sich gar nichts mehr tut, erlöst der Griff zum Netzschalter Sie von dem Übel.

Franz Dreismann,RH

Ungefähr ist nicht genau genug



BCD-Arithmetik auf dem C64

Auf der Benutzeroberfläche arbeitet unsere BCD-Arithmetik mit Textvariablen (Strings), wobei vorausgesetzt wird, daß diese Strings nur Ziffern (natürlich auch Dezimalpunkt und Negativ-Zeichen) enthalten dürfen. Im Prinzip sagen Sie dann: C$=A$-B$ (das stimmt zwar nicht ganz, aber zu der genauen Syntax-Beschreibung kommen wir noch).

Der Vorteil liegt in der Berechnung bis 250 Stellen (wenn Sie wollen, auch nur Nachkommastellen). Damit wird zum einen der Wertebereich des Interpreters bei weitem überschritten (250 Stellen gegenüber 39 Stellen), zum anderen wird innerhalb des zulässigen Wertebereichs auch wirklich genau gerechnet.

Demonstratives

Wenn Sie das Programm innerhalb von INPUT 64 geladen haben, können Sie sich erst einmal (mit f3) einige „Ungereimtheiten" des BASIC-Interpreters ansehen.

Auf der Rechenseite (mit f5 erreichbar) können Sie selbst Berechnungen ausführen. Die Ein- und Ausgabe ist hier nur aus optischen Gründen eingeschränkt. Außerhalb von INPUT 64 stehen Ihnen selbstverständlich alle 250 Stellen zur Verfügung.

Daß es mit der Rechengenauigkeit des BASIC-Interpreters nicht weit her ist, hat sich inzwischen wohl rumgesprochen. Auch der Wertebereich, in dem der C64 überhaupt rechnet, kann bei einigen Berechnungen einfach gesprengt werden. Sie kennen sicherlich die Fehlermeldung "Overflow Error". Zwar kann man bei den meisten Anwendungen auf ein genaues Ergebnis verzichten, wollen Sie aber sicher sein, daß eine Zuweisung auch korrekt ist, können Sie um den BASIC-Interpreter eigentlich nur einen großen Bogen machen. Ja gut, werden Sie sagen, aber was dann? Wir wissen nicht, was Commodore empfiehlt, wir empfehlen Ihnen unsere BCD-Arithmetik.

Wir bieten Ihnen insgesamt drei unterschiedliche Versionen zum Überspielen auf Ihren Datenträger an. Der SAVE-Befehl CTRL + S ist nur in dem Menüpunkt „Parameter einstellen" (mit f1 erreichbar) aktiv. Hier können Sie auch einstellen, welche Version Sie sichern wollen.

Einheitliches

Jede Version besteht aus fünf verschiedenen Rechenroutinen, vier für die Grundrechenarten und eine für die Rundung, wobei Rundung in diesem Fall nicht das mathematische Auf- oder Abrunden, sondern ein einfaches Abschneiden der überzähligen Stellen bedeutet.

Die Operatoren müssen vor der Berechnung in String-Variablen abgelegt werden. Das String-Format der Zah-

SYS-Adressen	Version 1	Version 2	Version 3
PRINT AT	2093	2093	entfällt
INLINE	2096	2096	entfällt
Addition	2816	2816	49152
Multiplikation	2836	2836	49172
Division	2846	2846	49182
Rundung	2856	2856	49192

Tabelle 1: Aufruf-Adressen für SYS-Befehle

len ist prinzipiell das gleiche, wie es auch der BASIC-Interpreter verwendet. Also: eventuell ein „-", dann Vorkommastellen, Dezimalpunkt, Nachkommastellen, aber keine Exponentenangabe! Der Raum vor der ersten Vorkommastelle darf dabei beliebig mit Leerzeichen aufgefüllt sein, was für die formatierte Zahlenausgabe sehr nützlich sein kann. Führende Nullen können wie üblich auch weggelassen werden.

Zu beachten ist auch, daß das laufende Programm nicht mehr wie gewohnt bei Fehlern abgebrochen wird, sondern es wird im Fehler-Flag ein Fehlercode übergeben, den der Benutzer dann selbst auswerten kann.

Sollten Ergebnisse mit mehr als 250 Stellen auftreten, so werden zunächst Nachkommastellen abgeschnitten (Fehler-Flag=1), erst dann wird ein Overflow-Error (Fehler-Flag=3) gegeben. Das Fehler-Flag sollten Sie grundsätzlich nach jeder Operation abfragen.

Unterschiedliches

Die Versionen unterscheiden sich in den SYS-Adressen (siehe Tabelle 1) und in den jeweils beanspruchten Speicherbereichen (siehe Tabelle 2) sowie in der Tatsache, daß wir in den beiden Versionen, die am BASIC-Anfang liegen, die Funktionen PRINT AT und INLINE mit eingebunden haben.

Alle Versionen benötigen einen Speicherbereich für das eigentliche Programm und einen Puffer für die String-Ablage.

Nur die Version 1 bleibt vollständig mit beiden Speicherbereichen im BASIC-Speicher. Hier wird der String-Puffer von dem verfügbaren BASIC-String-Bereich abgezweigt. Nach jeder Berechnung wird der Speicherbereich zwar wieder freigegeben, da dieser Puffer aber bei jedem Aufruf neu angefordert werden muß, können Sie — bei umfangreichen sonstigen (normalen) String-Eingaben — eventuell den insgesamt verfügbaren Speicherbereich sprengen.

Die Version 2 legt ihren Programmspeicher zwar auch an den BASIC-Anfang, der Puffer wird aber in den Bereich oberhalb(!) von SuperTape gelegt. Wenn Sie also keine weiteren Routinen in dem Bereich oberhalb von 49152 benötigen, ist diese Version sicherlich vorzuziehen.

Um die Versionen 1 und 2 fest in Ihr BASIC-Programm einzubinden, müssen Sie vor dem Sichern den BASIC-Anfang mit POKE 44,8 wieder auf den normalen Anfangswert setzen.

Die Version 3 läßt hingegen den ganzen BASIC-Bereich in Frieden. Allerdings können Sie zum einen SuperTape nicht mehr benutzen, und zum anderen kann diese Version nicht an Ihr BASIC-Programm angebunden werden. Sie müssen also erst die Version 3 laden, mit RUN anstarten und initialisieren, und danach als zweiten Schritt Ihr BASIC-Programm nachladen und gegebenenfalls starten.

Syntaktisches

Für die vier Grundrechenarten gilt die folgende Syntax:
SYS AD,A$,B$,C$,FL
oder
SYS AD,A$,B$,C$,FL,N
Die einzelnen Variablen haben folgende Bedeutung:

AD = Aufrufadresse, unterschiedlich je nach gewünschter Operation und Version (siehe Tabelle 1)
A$ = erster Operator
B$ = zweiter Operator
C$ = Ergebnis-String
FL = Fehler-Flag (nach der erfolgten Verknüpfung wird in dieser Variable ein Wert abgelegt)

Speicherbelegung	Version 1	Version 2	Version 3
Programm-Anfang	2049	2049	49152
Programm-Ende	5632	5632	52095
Daten-Anfang	variabel	51200	52096
Daten-Ende	variabel	52000	52863
BASIC-Anfang	5633	5633	2049

Tabelle 2: Speicherbelegung der unterschiedlichen Versionen

Die Ungenauigkeit des BASIC-Interpreters

Der BASIC-Interpreter legt Werte von Fließkomma-Variablen (im Gegensatz zu Strings) in kodierter und komprimierter Form ab. Hierbei wird für jede Zahl nur ein Bereich von fünf Bytes reserviert. In diesen wenigen Bytes müssen die Mantisse, der Exponent sowie auch noch negative Vorzeichen mit kodiert werden. Damit ergibt sich eine theoretische Genauigkeit von acht Nachkommastellen (1). Aufgrund der Trennung von Mantisse und Exponent kann zwar mit einem Wertebereich von absolut $1.7 * 10^{38}$ bis $2.9 * 10^{-39}$ gerechnet werden, nur, addieren Sie zum Beispiel eine sehr große und eine sehr kleine Zahl, wird der kleine Wert schlicht unterschlagen, da nicht mehr als insgesamt neun Ziffern dargestellt werden können.

Die Werte bedeuten:
0 = kein Fehler
1 = es wurde gerundet
2 = falsches Format, Rechnung wurde nicht durchgeführt
3 = Ergebnis zu groß (Overflow)
4 = Division durch 0
N = Nachkommastellen (Ist optional, muß also nicht angegeben werden. Falls doch, können Sie hier bestimmen, wieviel Nachkommastellen das Ergebnis haben soll.)

Die Nachkommastellen können Sie aber auch mit einer separaten Funktion beeinflussen. Die Syntax hierfür lautet dann:

SYS AD,A$,FL,N

Die Variablen bedeuten hier:
AD = Aufrufadresse (siehe Tabelle 1)
A$ = String, der gerundet werden soll
FL = Fehler-Flag (hier können nur die Werte 0,1,und 2 zurückgegeben werden)
N = Nachkommastellen (hier ist die Variable natürlich nicht optional)

Zusätzliches

Bei den Versionen 1 und 2, also *Versionen, die am BASIC-Anfang liegen*, haben wir zusätzlich die Funktionen PRINT AT und INLINE eingebunden. Die genauen Beschreibungen entnehmen Sie bitte dem Beiheft der Ausgabe 11/86. An dieser Stelle nur eine kurze Aufstellung der Parameter.

Syntax für PRINT AT:

SYS AD,Z,S,T$

Die Variablen haben folgende Bedeutung:
AD = Aufrufadresse (s. Tabelle 1)
Z = Zeilennummer
S = *Spaltennummer*
T$ = Ausgabe-String

Syntax für INLINE:

SYS AD,Z,S,L,DF$,ZI$,FL

Die Belegung der Variablen:
AD = Aufrufadresse (s. Tabelle 1)
Z = Zeilennummer
S = Spaltennummer
L = Länge der Eingabe (maximal)
DF$ = Definitions-String (enthält alle bei der Eingabe erlaubten Zeichen)
ZI$ = Ziel-String (hier wird die Eingabe übergeben)
FL = Ereignis-Flag (hier wird ein Wert übergeben, der von der Beendigung der Eingabe abhängt)
Die Werte bedeuten:

0	= RETURN
1	= CRSR-DOWN
-1	= CRSR-UP
-117	= f6-Taste
-118	= f4-Taste
-119	= f2-Taste
-120	= f7-Taste
-121	= f5-Taste
-122	= f3-Taste
-123	= f1-Taste

Anregendes

Sie können selbstverständlich die Einschränkungen auf die vier Grundrechenarten der BCD-Arithmetik umgehen, indem Sie einen oder beide Operanden vorher mit dem Interpreter auf der Basis von Real-Variablen berechnen und vor dem Aufruf in einen String (mit STR$) umwandeln.

Allerdings kann bei der Wandlung eines BCD-Strings in eine „normale" Zahlenvariable (mit VAL) der Interpreter aussteigen, wenn der Wertebereich den erlaubten Bereich für Real-Variablen übersteigt (OVERFLOW ERROR).

Die zehn Finger des 6510

Der Mikroprozessor 6502 (und damit auch der 6510 im C64) kann mit sogenannten "Binär-Codierten Dezimalen", BCDs, arbeiten (2). Dabei wird jedes Nibble (das sind vier Bits, also ein halbes Byte) als eine Dezimalziffer aufgefaßt. Da in vier Bits 16 unterschiedliche Informationen (2^4=16) kodiert werden können, bei der BCD-Darstellung aber nur 10 Unterscheidungen gebraucht werden, bleiben 6 unbenutzte Zustände übrig, was normalerweise zu Fehlern bei Überträgen zwischen den Nibbles führt. Durch Setzen des Dezimal-Flags im Status-Register (Maschinensprache-Befehl: SED) kann man dem Prozessor mitteilen, daß er sich selbst um diese Überträge kümmern soll. Der Additions-Befehl (ADC) und der Subtraktions-Befehl (SBC) können danach bedenkenlos weiterverwendet werden. Das Verdoppeln von Zahlen wird einfach durch Addition mit sich selbst realisiert. Lediglich beim Halbieren (mittels ROR oder LSR) ist besondere Vorsicht geboten. Aufpassen müssen Sie auch, wenn beispielsweise in Schleifenkonstruktionen "normale" Rechnungen vorkommen. Vorher muß auf jeden Fall das Dezimal-Flag wieder zurückgesetzt werden (CLD).

Es macht schon Spaß, mit großen Zahlen zu rechnen. Sie könnten zum Beispiel einmal in einer Schleife richtig große Zweierpotenzen berechnen. Die größte mit der BCD-Arithmetik (genau) darstellbare Zahl ist dann $2^{*}2^{847}$. Ersparen Sie es uns bitte, diese Zahl hier hinschreiben zu müssen.

Martin Gebhardt/WM

Literaturhinweise:

(1) Lothar Englisch: Das Maschinensprachebuch für Fortgeschrittene zum Commodore 64; Data Becker; Düsseldorf 1984; ISBN 3-89011-022-3

(2) Christian Persson: 6502/65C02 Maschinensprache; Heise-Verlag; Hannover 1983; ISBN 3-922705-20-0

c't-Platinen

Unsere Schwesterzeitschrift c't hält ein umfangreiches Platinen-Angebot für Sie bereit. Eine aktuelle Übersicht wird in jeder c't-Ausgabe veröffentlicht. Für INPUT-64-Anwender sind insbesondere die folgenden zwei Platinen interessant:

C-64-Sound-Sampler
(Die Karte für INPUT SAM)
Bestell-Nr.: 860972dB
zum Preis von 35,- DM

C-64-EPROM-Bank
(Die Karte für die EPROM-Version von INPUT-BASIC)
Bestell-Nr.: 8412112dB
zum Preis von 18,- DM

Bei beiden Karten handelt es sich um doppelseitige Leerplatinen mit Bestückungsaufdruck. Die Preise verstehen sich zuzüglich 3,- DM für Porto und Verpackung. Lieferung nur gegen Vorauszahlung.

Bestelladresse:
Verlag Heinz Heise GmbH,
Postfach 610407,
3000 Hannover 61

Zahlensalat

Chiffrierung in der Rätsel-Ecke

Wenn Geheimdienste einen Funkspruch der gegnerischen Seite abfangen, fängt die Arbeit erst richtig an. Die empfangenen Daten sind nämlich in der Regel chiffriert und ergeben beim ersten Lesen keinen Sinn. (Über den Sinn nach einer Dechiffrierung wollen wir uns hier nicht weiter auslassen.) Geheimdienste werden bei dieser Arbeit von Computern unterstützt. Komplexe Programme versuchen, auf Großrechnern die vormals mühselige Kleinarbeit von vielen Mathematikern zu übernehmen.

Selbstverständlich verfügen Geheimdienste über sehr viel komplexere Chiffrierungs-Schlüssel, als wir sie hier in unserem Spiel verwenden. Uns geht es bei dieser Rätselaufgabe natürlich nicht um ein universales Dechiffrierungs-Programm, sondern . . .

James in Nöten

James (nicht der Buttler!) hatte durch seinen besten Mitarbeiter Dr. Zufall den Klartext für die ersten beiden Funksprüche erhalten, und staunte nicht schlecht, als er feststellen mußte, daß beide Funksprüche bei unterschiedlichen Codes den gleichen Inhalt hatten.

Als er nun auch den dritten und vierten Funkspruch zugespielt bekommen und aus gewöhnlich gut unterrichteten Kreisen erfahren hatte, daß auch diese inhaltlich identisch seien und die gleiche Verschlüsselungstechnik wie bei den ersten beiden Funksprüchen verwendet wurde, rief er kurz entschlossen in der INPUT-Redaktion an und bat um die Mithilfe der 100.000 INPUT-Leser.

Wir kommen hiermit dieser Bitte nach und veröffentlichen die Kodierungen aller vier Funksprüche. Damit sich Ihre Arbeit auch lohnt, werden wir unter den richtigen Lösungen (wir suchen nur den Klartext!) fünf Bücher auslosen. Die Chance, ein Buch zu gewinnen, haben Sie aber nur, wenn Ihre Lösung bis Freitag, den 3. April 1987, hier bei uns eingetroffen ist.

Hilfestellungen

Damit Sie die einzelnen Codes des dritten und vierten Funkspruchs nicht abzutippen brauchen, können Sie innerhalb des Programms Rätsel-Ecke ein Rahmen-Programm für Ihre Lösungsversuche (wie gewohnt mit CTRL + S) auf Ihren Datenträger

Erster abgefangener Funkspruch:

```
 67, 116,  91, 207, 142,  72,
140,  87, 127, 212, 137,  68,
130, 140, 127, 123, 129,  27,
124, 176, 123, 103,  90, 208,
126,  27, 140, 163, 131,  24,
145,  44,  90,  94, 136, 159,
137, 205, 131,  79, 142, 134,
123, 116, 135, 242, 140, 146,
137, 234, 128,  67, 136,  88,
259,  13,  90, 244, 127, 177,
 141,  21, 127,   8, 131, 146,
254,  52,
```

Die Chiffrierung bedeutet:
„Diese Information wird abgehört!"

Tabelle 1:
Der erste geknackte Code

Zweiter abgefangener Funkspruch:

67, 70, 68, 158, 119, 27,
117, 133, 104, 40, 114, 217,
107, 128, 104, 247, 106, 212,
101, 76, 100, 165, 67, 101,
103, 153, 117, 1, 108, 234,
122, 49, 67, 143, 113, 162,
114, 246, 108, 111, 119, 180,
100, 144, 112, 163, 117, 109,
114, 244, 105, 244, 113, 110,
236, 82, 67, 161, 104, 130,
118, 91, 104, 80, 108, 135,
231, 229,

Auch diese Chiffrierung bedeutet:
„Diese Information wird abgehört!"

Tabelle 2:
Der zweite geknackte Code

Dritter abgefangener Funkspruch:

63, 94, 80, 109, 131, 27,
122, 189, 114, 14, 112, 149,
125, 131, 122, 135, 116, 23,
118, 188, 79, 239, 131, 236,
130, 31, 120, 147, 79, 101,
116, 237, 115, 60, 126, 40,
242, 220, 79, 103, 129, 99,
116, 91, 243, 31, 79, 203,
80, 229, 118, 101, 125, 73,
132, 211, 131, 65, 119, 183,
114, 30, 240, 237,

Dieser Text wird gesucht.

Tabelle 3:
Der erste unbekannte Code

überspielen. Außerhalb von INPUT 64 können Sie dann in diesem Programm frei editieren und damit unterschiedliche Lösungsansätze ausprobieren.

Selbstverständlich sind alle vier Funksprüche mit dem gleichen Programm verschlüsselt worden und demzufolge auch mit einem Programm wieder zu entschlüsseln.

Wir möchten Ihnen an dieser Stelle noch einige Tips geben. (Wer sich ohne diese durch den Zahlensalat wühlen möchte, sollte gleich bei der folgenden Überschrift weiterlesen.)

Wenn Sie das Rahmenprogramm analysieren, werden Sie das „Geheimnis" der ersten Zahl sofort erkennen.

Es könnte hilfreich sein, die Anzahl der Codes und die Anzahl der Klartext-Zeichen zu untersuchen.

Wenn unterschiedliche Code-Folgen den gleichen Inhalt haben können, muß irgendwo genau dieser Unterschied auch (verschlüsselt) enthalten sein.

Jetzt sollten wir aber aufhören, sonst geht der Reiz des Spiels verloren.

Einer gegen alle

Wir möchten dieses Verschlüsselungs-Spiel gerne zu einer neuen Serie in INPUT 64 werden lassen. Hiermit fordern wir Sie also auf, eigene Chiffrierungs-Programme zu entwikkeln und diese zur Verfügung zu stellen. Wir erwarten dabei ein Programm, das sowohl die Chiffrierung als auch die Dechiffrierung leistet.

Vierter abgefangener Funkspruch:

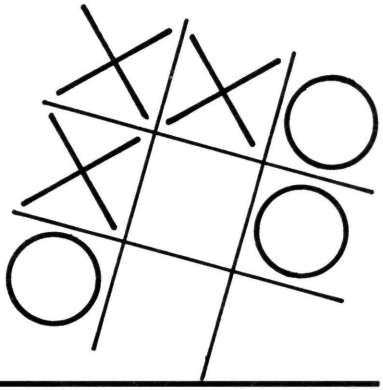
63 88, 77, 191, 128, 210,
119, 160, 111, 174, 109, 16,
122, 79, 119, 124, 113, 243,
115, 152, 76, 23, 128, 96,
127, 158, 117, 185, 76, 164,
113, 189, 112, 135, 123, 7,
239, 68, 76, 76, 126, 42,
113, 152, 240, 182, 76, 84,
77, 206, 115, 221, 122, 233,
129, 212, 128, 162, 116, 98,
111, 218, 237, 220,

Dieser Text wird gesucht, er ist identisch mit dem dritten abgehörten Funkspruch.

Tabelle 4:
Der zweite unbekannte Code

Sollte es keinem INPUT-Leser gelingen, innerhalb einer bestimmten Frist den Code zu knacken, bekommt der Programm-Entwickler ein INPUT-64-Jahresabonnement. Ansonsten werden unter den richtigen Einsendungen, wie auch diesmal schon, fünf Bücher ausgelost. Der Programmentwickler geht in diesem Fall leider leer aus. Es spielt also einer gegen alle. Wir können uns vorstellen, daß dieses für beide Seiten ein spannender Wettkampf werden kann. WM

Tic-Tac oder Tak-Tik?



Das selbstlernende Spiel

Mit dem alten, bekannten Spiel Tic-Tac-Toe soll demonstriert werden, daß nicht immer nach der „X Schritte im voraus denken"—Methode gespielt werden muß, um achtbare Ergebnisse zu erzielen.

Die Spielregeln sind allgemein bekannt. Auf einem Drei-mal-drei-Spielfeld setzen zwei Spieler abwechselnd ihre Figuren. Gewonnen hat derjenige, der als erster drei gleiche Figuren (x oder o) nebeneinander hat (waagerecht, senkrecht oder diagonal). Diese Stellung nennen wir im folgenden „Mühle". Der Gegner muß also eine solche Mühle des Gegenspielers verhindern und sich selber eine solche aufbauen.

„Altbekanntes Spiel, schon 1000mal veröffentlicht", werden Sie sagen. Ja, ja, wissen wir. Dennoch hat diese Spielstrategie (gebenüber den herkömmlichen, dem Mini-Max-Algorithmus, wenn's sein muß, auch mit Alpha-Beta-Schnitt), einen gewissen Reiz. Werfen Sie das Programm auch nicht gleich in den Papierkorb, wenn Sie gegen den Computer immer gewinnen. Bei dieser Strategie, die hier angewandt wird, geht es vielmehr darum, daß der Computer (sprich: Programm) erst sehr dumm ist. Das Gehirn des Computers ist leer. Doch mit der Zeit lernt er aus seinen Fehlern und versucht diese nicht mehr zu machen.

Die Strategie . . .

Nun zum Programm selber. Der Computer ist, wie schon erwähnt, zu Beginn des Spiels sehr dumm. Wir lassen ihn (in Gedanken) einfach mal anfangen. Er setzt seine Figur (x) auf das erste, freie Feld (links oben in der Ecke). Sie, als der Gegenspieler, setzen auf das Feld rechts daneben. Der Computer setzt wieder auf das nächste freie Feld (rechts oben in der Ekke). Sie wiederum setzen auf das rmittlere Feld (Spielfeldmitte). Der Rechner links daneben. Jetzt haben Sie die Möglichkeit, Ihre Mühle zu vervollständigen, indem Sie auf das mittlere Feld der unteren Reihe setzen. Sie haben gewonnen, der Computer verloren.

Aha, denkt der Computer, das war nichts. Beim nächsten Spiel weiß er, daß er diesen Zug nicht wieder machen darf, wenn er nicht verlieren will, und „sagt zu sich": „mit dem ersten Feld nicht wieder anfangen. Wir nehmen einfach das nächste".

Das geht jetzt immer so weiter. Bei jedem neuen Spiel wird also nachgesehen, ob der Zug schon mal verwendet wurde und zu einem guten oder schlechten Ergebnis führte. Und schon sind wir beim Thema. Um solch eine Spielstrategie in einem Spiel zu programmieren, muß man sich natürlich viele Gedanken machen. Als erstes brauchen wir zwei „Kisten", in die wir einmal die guten und einmal die schlechten Züge ablegen. Bei jedem Zug muß in den beiden „Kisten" nachgesehen werden, ob der Zug, der als nächstes ausgeführt werden soll, schon mal gespielt wurde. Wird er in der „Kiste" für die guten Züge gefunden (da muß als erstes gesucht werden), wird er ohne zu zögern ausgeführt. Wird er nicht gefunden, muß in der schlechten „Kiste" gesucht werden. Wird er da gefunden, darf er auf keinen Fall ausgeführt werden.

. . . kann auch eine . . .

Dazu müssen folgende Vorüberlegungen angestellt werden: in welchem Format lege ich die einzelnen Züge ab, wie lege ich die Züge so ab, daß sich die Zahlen nicht wiederholen, wie lege ich sie ab, damit sie schnell wiedergefunden werden können? Kommen wir zum Problem des Formats und der Ablage.

Bei einem Spielfeld mit drei mal drei Kästchen bietet sich die Möglichkeit an, die einzelnen Kästchen mit Zahlen aus der Reihe der Dreierpotenzen zu belegen. In der ersten Reihe von links nach recht ergeben sich also folgende Zahlen: 1, 3, 9; in der zweiten Reihe: 27, 81, 243 und so weiter. Nehmen wir also noch mal an, der Rechner besetzt Feld 1. Eingetragen wird eine 1. Der Spieler belegt Feld 2. Jetzt wird

aber keine 3 eingetragen, sondern die 3 vom Spieler wird mit 2 multipliziert (3*2), das Ergebnis ist 6. Zu dieser 6 wird die 1 des Computerzuges hinzuaddiert, und wir haben eine 7. In unserer Zahlenreihe stehen die beiden Zahlen: 1, 7.

Beim nächsten Zug des Rechners (er wählt Feld 3) wird die 9 wieder zu der zuletzt berechneten Zahl addiert. Das Ergebnis ist 16. Wie beim oben erwähnten Beispiel setzen Sie jetzt wieder auf Feld 5, das entspricht der Dreierpotenz 81. Diese wird wieder verdoppelt, das ergibt 162. Dazu wird die 16 addiert, und wir haben die nächste Zahl, die wir eintragen können, eine 178. Der Rechner wiederum belegt Feld 4. Wir addieren eine 27. Das ergibt die Zahl 205. Sie sind als nächstes dran und setzen auf das Feld 8. In diesem Feld steht die Zahl 2187. Diese wird wieder verdoppelt (4374) und zur letzten Zahl addiert. Es ergibt sich also die Zahl 4579. Das ist auch die letzte Zahl, die eingetragen wird, denn Sie haben das Spiel ja gewonnen. Diese Zahlenreihe aus sechs Zahlen sagt die genaue Zugfolge dieses einen Spiels aus. Eine andere Zugfolge würde auch eine andere Zahlenreihe ergeben. Der Computer kann sich so ganz einfach jeden Zug merken.

Die andere Frage, die sich stellt, ist die: wo trage ich die Zahlenreihe ein? Dazu wird ein Feld mit neun einzelnen Feldern dimensioniert, zum Beispiel: TP(X). TP(1)=1, TP(2)=7, TP(3)=16, TP(4)=178, und so weiter.

Kommen wir zu dem Problem, wie wir die einzelnen Züge schnell wiederfinden. In BASIC geht das alles zwar nicht so schnell wie in Maschinensprache, aber immerhin. Nehmen wir uns dazu der guten und der schlechten „Kisten" an. Wir dimensionieren zwei zweidimensionale Felder mit DIMG(150,9) und DIMS(150,9), wobei das „G" für die „Kiste" der guten und das „S" für die „Kiste" der schlechten Züge steht. Die erste Zahl, die 150, sagt aus, daß bis zu 150 Spielzüge für Gut und 150 Spielzüge für Schlecht abgelegt werden können: insgesamt also 300 Züge. Die zweite Zahl, die 9, sagt aus, daß in einem Spiel bis zu 9 Züge möglich sind. Werden es weniger als 9 Züge, wird der Rest mit Nullen aufgefüllt. Nachdem nun entschieden ist, ob der Computer verloren oder gewonnen hat, werden die Zahlen aus dem temporären Feld (TP(X)) in eines der Felder für Gut oder Schlecht abgelegt, und das Feld für die Zwischenspeicherung ist wieder frei, um einen neuen Zug aufzunehmen.

. . . andere sein

Bei jedem neuen Spielzug muß der Computer also ein den dimensionierten Feldern S(X,Y) und G(X,Y) nachsehen, ob solch ein Spielzug schon existiert und ob er gut oder schlecht ist. Je nachdem ob und wenn, wo er gefunden wurde, muß er dementsprechend reagieren. Findet er den nächsten Zug in keinem der beiden Felder, nimmt er einfach den nächstmöglichen. Ob er dadurch gewinnt, sei dahingestellt. Auch ist nicht gesagt, ob der Computer nach dreihundert möglichen Spielen schon so schlau geworden ist, daß er immer gewinnt; das kann auch erst nach fünfhundert oder tausend Spielen eintreten. Jedenfalls lernt er von Spiel zu Spiel dazu und wird immer schlauer. Speichern Sie sich dieses Programm wie üblich mit CTRL+S auf Ihren eigenen Datenträger ab, und versuchen Sie einmal, in diese Strategie einzusteigen. Vielleicht gelingt Ihnen hier und da eine Verbesserung des Programms. Mich würde es freuen! kfp

Lernen im Dialog

Englische GRAMmatik

In der eigenen Muttersprache hat man das passende Verhältniswort im Gefühl, fremdsprachliche Präpositionen muß man üben — beispielweise mit diesem Programm.

Im vierten Teil der Serie geht es um die Auswahl zwischen den englischen Präpositionen about, above, after, along, at, before, below, by, down, for, from, in, into, off, on, onto, over, past, through, to, under, up, until, within.

Anders ausgedrückt: die
— Prepositions of place
— Prepositions of movement
— Prepositions of time

Die richtigen Eingaben müssen jeweils in die Textlücken der Beispielsätze eingegeben werden (mit RETURN anschließen). Nach jeder Eingabe können Sie entweder eine der auf dem Bildschirm gezeigten Möglichkeiten wählen oder mit einer beliebigen anderen Taste mit der nächsten Frage fortfahren. Alles weitere erfahren Sie im Programm — also: Laden und Lernen! JS

Hinweise zur Bedienung

Bitte entfernen Sie eventuell vorhandene Steckmodule. Schalten Sie vor dem Laden von INPUT 64 Ihren Rechner einmal kurz aus. Geben Sie nun zum Laden der Kassette

AD und RETURN

beziehungsweise bei der Diskette

LOAD"INPUT",8,1 und RETURN

ein. Alles weitere geschieht von selbst.

Sollten Sie ein Laufwerk haben, das zusammen mit dem Schnellader der Diskettenversion Schwierigkeiten macht, geben Sie bitte ein

LOAD"LADER*",8,1 und RETURN.

Nach der Titelgrafik springt das Programm in das Inhaltsverzeichnis des Magazins. Dieses können Sie nun mit SPACE (Leertaste) durchblättern. Mit RETURN wird das angezeigte Programm ausgewählt und geladen. Im Fenster unten rechts erhalten Kassetten-Besitzer weitere Hinweise („Bitte Band zurückspulen" und so weiter ...).

Haben Sie bei der Auswahl eines Programms eventuell nicht weit genug zurückgespult und es wurde nicht gefunden, spulen Sie bis zum Bandanfang zurück.

Auf der zweiten Kassetten-Seite befindet sich eine Sicherheitskopie. Sollten Sie eventuell mit einem Programm *Ladeschwierigkeiten haben, versuchen* Sie es auf der zweiten Seite. Führt auch dies nicht zum Erfolg, lesen Sie bitte die entsprechenden Hinweise im Kapitel „Bei Ladeproblemen"!

Neben der Programmauswahl mit SPACE und dem Ladebefehl mit RETURN (im Inhaltsverzeichnis) werden die übrigen 'System-Befehle' mit der Kombination aus CTRL-Taste und einem Buchstaben eingegeben. Sie brauchen sich eigentlich nur CTRL und H zu merken (Aufruf der Hilfsseite), denn dort erscheinen die jeweils möglichen 'System-Befehle'. Nicht immer sind alle Optionen möglich (eventuell werden Sie zu Beginn des Programms auf Einschränkungen hingewiesen). Hier nun alle INPUT-64-Systembefehle:

CTRL und Q

Sie kürzen die Titelgrafik ab; INPUT 64 geht dann sofort ins Inhaltsverzeichnis.

CTRL und H

Es wird ein Hilfsfenster angezeigt, auf dem alle verfügbaren Befehle aufgeführt sind.

CTRL und I

Sie verlassen das Programm und kehren in das Inhaltsverzeichnis zurück.

CTRL und F

Ändert die Farbe des Bildschirm-Hintergrundes (auch im Inhaltsverzeichnis erreichbar)

CTRL und R

Ändert die Rahmenfarbe (auch im Inhaltsverzeichnis erreichbar).

CTRL und B

Sie erhalten einen Bildschirmausdruck — natürlich nicht von Grafikseiten oder Sprites! Angepaßt ist diese Hardcopy für Commodore-Drucker und kompatible Geräte. Das Programm wählt automatisch die richtige Geräteadresse (4,5 oder 6) aus.

CTRL und S

Wenn das Programm zum Sichern vorgesehen ist, erscheinen weitere Hilfsfenster. Sie haben die Wahl, ob Sie:

im Commodore-Format C
im SuperTape-Format S
auf Diskette D

sichern wollen. Beachten Sie bitte, daß Sie die Programme von Ihrem Datenträger immer als normale BASIC-Programme mit LOAD"NAME",1 bzw. LOAD"NAME",8 laden müssen. Wenn Sie das Programm im SuperTape-Format aus INPUT 64 abgespeichert haben, müssen Sie vor dem Laden selbstverständlich Super-Tape in Ihren Rechner geladen und initialisiert haben. (SuperTape DII haben wir in der Ausgabe 4/85 veröffentlicht.) Außerdem wird in diesem Fenster die Programmlänge in Blöcken angegeben. Kassetten-Benutzer können diese Disketten-Blockzahl nach folgender Faustregel umrechnen: Im Commodore-Format werden pro Minute neun Blöcke abgespeichert, SuperTape schreibt die gleiche Anzahl von Blöcken in cirka sechs Sekunden aufs Band.

Bei Ladeproblemen

Diskette: Bei nicht normgerecht justiertem Schreib-/Lesekopf oder bei bestimmten Serien wenig verbreiteter Laufwerke (1570) kann es vorkommen, daß das im INPUT-Betriebssystem eingebaute Schnelladeverfahren nicht funktioniert. Eine mögliche Fehlerursache ist ein zu geringer Abstand zwischen Floppy und Monitor/Fernseher. Das Magazin läßt sich auch im Normalverfahren laden, eventuell lohnt sich der Versuch:

LOAD"LADER",8,1

Sollte auch dies nicht zum Erfolg führen, senden Sie bitte die Diskette mit einem kurzen Vermerk über die Art des Fehlers und die verwendete Gerätekonstellation an den Verlag (Adresse siehe Impressum).

Kassette: Schimpfen Sie nicht auf uns, die Bänder sind normgerecht nach dem neusten technischen Stand aufgezeichnet und sorgfältig geprüft. Sondern: Reinigen Sie zuerst Tonköpfe und Bandführung Ihres Kassettenrecorders. Die genaue Vorgehensweise ist im Handbuch der Datassette beschrieben. Führt auch dies nicht zum Erfolg, ist wahrscheinlich der Tonkopf Ihres Gerätes verstellt. Dieser Fehler tritt leider auch bei fabrikneuen Geräten auf.

Wir haben deshalb ein Programm entwikkelt, mit dessen Hilfe Sie den Aufnahme-/Wieder gabekopf justieren können. Tippen Sie das Programm JUSTAGE ein und speichern Sie es ab. Dieses Programm wertet ein etwa 30 Sekunden langes Synchronisationssignal aus, das sich am Ende jeder Kassettenseite befindet. Starten Sie das JUSTAGE-Programm mit RUN, jetzt sollte die Meldung PRESS PLAY ON TAPE kommen, drücken Sie also die PLAY-Taste. Nach dem Drücken der Taste geht der Bildschirm zunächst wie immer aus. Wird das Synchro-Signal erreicht, wechselt die Bildschirmfarbe, und zwar — bei nicht total verstellter Spurlage — völlig regelmäßig etwa dreimal pro Sekunde. Liegt die Spur des Tonkopfes grob außerhalb der zulässigen Toleranzgrenzen, geschieht entweder nichts, oder die Farben wechseln unregelmäßig. Nehmen Sie jetzt einen kleinen Schraubenzieher und werfen Sie einen Blick auf Ihre Datassette. Über der REWIND-Taste befindet sich ein kleines Loch. Wenn Sie bei gedrückter PLAY-Taste durch dieses Loch schauen, sehen Sie den Kopf der Justierschraube für die Spurlage. Drehen Sie diese Einstellschraube. Aber Vorsicht: ganz langsam drehen, ohne dabei Druck auszuüben! Drehen Sie die Schraube nicht mehr als eine Umdrehung in jede Richtung. Nach etwas Ausprobieren wird der Bildschirm gleichmäßig die Farbe wechseln. Zur Feinabstimmung lassen Sie das Synchro-Signal noch einmal von Anfang an laufen. Die Schraube jetzt nach links drehen, bis der Farbwechsel unregelmäßig wird. Diese Stellung genau merken, und die Schraube jetzt langsam wieder nach rechts drehen: Der Farbwechsel wird zunächst gleichmäßig, bei weiterem Drehen wieder unregelmäßig. Merken Sie sich auch diese Stellung, und drehen Sie die Schraube nun in Mittelstellung, das heißt zwischen die beiden Randstellungen. Denken Sie daran, daß während der Einstellung kein Druck auf den Schraubenkopf ausgeübt werden darf! Der Tonkopf Ihres Recorders ist jetzt justiert.

Sollte sich auch nach dieser Einstellung INPUT 64 nicht laden lassen, erhalten Sie von uns eine Ersatzkassette. Schicken Sie bitte die defekte Kassette mit einem entsprechenden Vermerk an den Verlag ein (Adresse siehe Impressum).

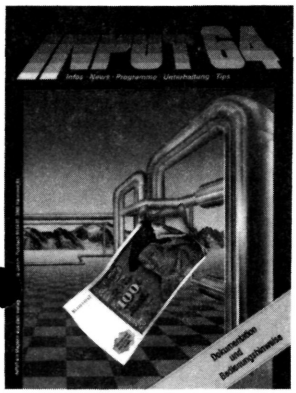
PS! In der Ausgabe 6/85 haben wir das Programm RECORDER-JUSTAGE veröffentlicht, das die Einstellung des Datenrecorders zum Kinderspiel macht.

Listing Justage

```
800 for i=49199to49410:read d:ps=ps+d:poke i,d:next
900 ifps<>24716thenprint"falsch abgetippt - fehler korrigieren!":end
950 print"o.k."
970 sys49338
1000 rem von 49199 bis 49410
1010 data173, 13,220,169,217,174,  4,220,172,  5,220,141, 14,220, 48, 44, 56
1020 data102, 88, 36, 89, 48, 12,144, 10,165, 88,133, 90,169,128,133, 88,133
1030 data 91,192,121,144,  4,224,115,176,  7,169,  0,133, 92, 56,176, 11,165
1040 data 92, 73,128,133, 92, 36, 92, 16, 19, 24,102, 88, 36, 89, 48, 12,144
1050 data 10,165, 88,133, 90,169,128,133, 88,133, 91,104,168,104,170,104, 64
1060 data 96, 36, 91, 16,252,132, 91,165, 90, 96,160,128,132, 89,165, 88,201
1070 data 22,208,250,132, 88,160, 10,132, 89,132, 91, 36, 91, 16,252,132, 91
1080 data165, 90,201, 22,208,226,136,208,241, 32,133,192,201, 22,240,249, 96
1090 data 32,147,252,120, 32, 23,248,165,  1, 41, 31,133,  1,133,192,169, 47
1100 data141, 20,  3,169,192,141, 21,  3,169,127,141, 13,220,169,144,141, 13
1110 data220,173, 17,208, 41,239,141, 17,208,169, 70,141,  4,220,169,129,141
1120 data  5,220, 88, 32,142,192,201, 42,208,249,173, 32,208, 41, 15,168,200
1130 data140, 32,208, 76,237,192,208, 76

ready.
```

Am 6. April 87 auf Kassette und Diskette an Ihrem Kiosk: INPUT 64, Ausgabe 4/87



Wir bringen unter anderem:

Wärmebedarfs-Berechnung

Ist Ihre Wohnung optimal beheizt und gut isoliert? Mit diesem Programm können Sie feststellen, welche Energiemengen für ein angenehmes Wohnklima verbraucht werden, und planen, mit welchen Umbaumaßnahmen wieviel Energie gespart werden kann.

Spekulator

Kämpfen Sie mit 200000 DM gegen einen Schuldenberg von 15 Millionen DM. Nur geschicktes Spekulieren mit Aktien, Immobilien und Gemälden kann Sie aus der Klemme befreien. Hazardeure setzen beim Roulett alles auf eine Kugel.

ReAssembler

Dieses Programm verwandelt Maschinen-Programme in INPUT-ASS kompatiblen Quellcode. Exotische oder nicht ganz eindeutige Umwandlungen, die Lage von Tabellen und so weiter werden interaktiv erfragt. Alle Anweisungen und eingegebene Label lassen sich als Tabellen zuladen und abspeichern.

INPUT-Assembler-Schule

Es geht weiter: Maschinen-Sprache des 6502 direkt am Rechner gelernt, geübt und verstanden.

und außerdem:

Spiele, Englische Grammatik, 64er-Tips . . .

c't – Magazin für Computertechnik

Ausgabe 4/87 – ab 19.3.1987 am Kiosk

Report: Kann man ein Menschengehirn „in Silizium" nachbauen? * Projekt: Preiswerter PC-Festplatten-Controller an Z80-Rechnern * Softare-Know-how: Computer-Viren — Kleinstprogramme „zerfressen" Datenbestände * Grundlagen: Mehr Speicher in PCs — EMS und Above Boards * Programm: „Bad-sector-Problem" in CP/M-Systemen entschärfen * u.v.a.m

elrad – Magazin für Elektronik

Ausgabe 3/87 – ab 23.2.1987 am Kiosk

Report: Layout — Software — An der Glotze kleben * Bauanleitung Audio: HiFi-Röhrenendstufe 2x60 W * Grundlagen: Berechnung von Transistorstufen * Bauanleitung Modellbau: Auto-Pilot * Die elrad-Laborblätter: Elektromechanische und elektronische Relais * Bauanleitung Meßtechnik: Sweep-Generator * Englisch für Elektroniker: Transformers * u.v.a.m

IMPRESSUM:

INPUT 64
Das elektronische Magazin

Verlag Heinz Heise GmbH
Bissendorfer Straße 8
3000 Hannover 61
Postfach 61 04 07
3000 Hannover 61
Telefon: (05 11) 53 52 - 0

Technische Anfragen:
nur dienstags von 9.00 — 16.30 Uhr

Postgiroamt Hannover, Konto-Nr. 93 05 - 308
(BLZ 250 100 30)
Kreissparkasse Hannover, Konto - Nr. 000 - 01 99 68
(BLZ 250 502 99)

Herausgeber: Christian Heise

Redaktion:
Christian Persson (Chefredakteur)
Ralph Hülsenbusch
Wolfgang Möhle
Karl-Friedrich Probst
Jürgen Seeger
Redaktionsassistent: Wolfgang Otto

Ständige Mitarbeiter:
Peter S. Berk
Irene Heinen
Peter Sager
Hajo Schulz
Eckart Steffens

Vertrieb: Anita Kreutzer

Grafische Gestaltung:
Wolfgang Ulber, Dirk Wollschläger

Herstellung: Heiner Niens

Lithografie:
Reprotechnik Hannover

Druck:
Leunismann GmbH, Hannover
CW Niemeyer, Hameln

Konfektionierung:
Lettershop Brendler, Hannover

Kassetten- und Diskettenherstellung:
SONOPRESS GMBH, Gütersloh

INPUT 64 erscheint monatlich.
Einpelpreise Kassette DM 16,80
Jahresabonnement Inland Kassette DM 140,—
Diskette DM 198,—
Einzelpreis Diskette DM 19,80

Redaktion, Abonnementverwaltung:
Verlag Heinz Heise GmbH
Postfach 61 04 07
3000 Hannover 61
Telefon: (05 11) 53 52 - 0

Abonnementverwaltung Österreich:
Evb-Verlag GmbH & Co KG
Abt. Zeitschriftenvertrieb
z. Hd. Frau Pekatschek
Amerlingstraße 1
A-1061 Wien
Jahresabonnement: Kassette DM 152,—
Diskette DM 210,—

Vertrieb (auch für Österreich, Niederlande, Luxemburg und Schweiz):
Verlagsunion Zeitschriften-Vertrieb
Postfach 57 07
D-6200 Wiesbaden
Telefon: (0 61 21) 2 66 - 0

Verantwortlich:
Christian Persson
Bissendorfer Straße 8
3000 Hannover 61

ISSN 0177 - 3771

Titelidee: **INPUT 64**
Titelillustration: Michael Thiele, Dortmund
Titel - Grafik und -Musik: Tim Pritlove, Fabian Rosenschein
Bertiebssystem: Hajo Schulz

hier abtrennen

INPUT 64-Abonnement Abruf-Coupon

Ja, übersenden Sie mir bis auf Widerruf alle künftigen INPUT64-Ausgaben ab Monat

_____

(Kündigung ist jederzeit mit Wirkung ab der jeweils übernächsten Ausgabe möglich. Überzahlte Abonnementsgebühren werden sofort anteilig erstattet.)

Das Jahresabonnement kostet: ○ auf Kassette DM 140,— inkl. Versandkosten und MwSt.

○ auf Diskette DM 198,— inkl. Versandkosten und MwSt.

(Bitte ankreuzen/Nichtzutreffendes streichen.)

Absender und Lieferanschrift

Bitte in jedes Feld nur einen Druckbuchstaben (ä = ae, ö = oe, ü = ue)

Vorname/Zuname

Beruf/Funktion

Straße/Nr.

PLZ Wohnort

_____

Datum/Unterschrift

Von meinem Recht zum schriftlichen Widerruf dieser Order innerhalb einer Woche habe ich Kenntnis genommen. Zur Wahrung der Frist genügt die rechtzeitige Absendung.

Unterschrift _____

Bitte beachten Sie, daß diese Bestellung nur dann bearbeitet werden kann, wenn beide Unterschriften eingetragen sind.

INPUT 64-Abonnement Abruf-Coupon

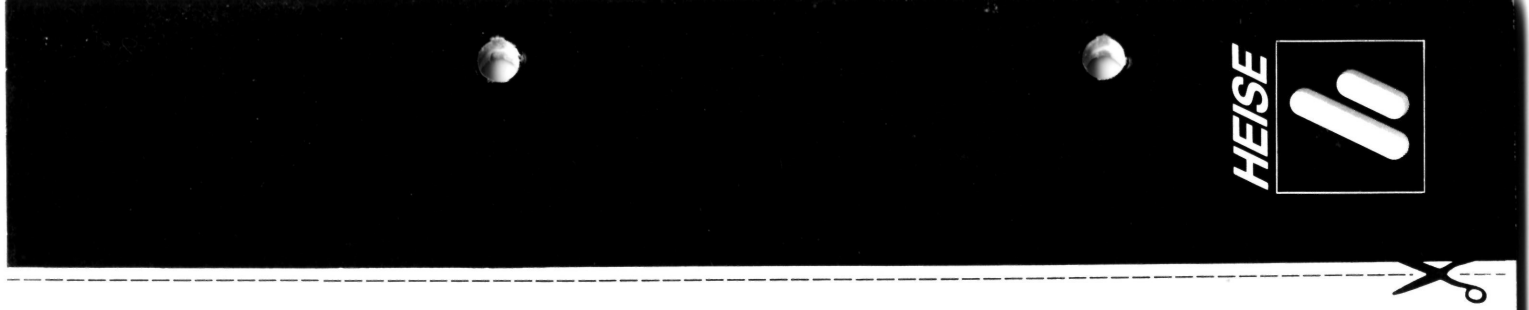
Ich wünsche Abbuchung der Abonnement-Gebühr von meinem nachstehenden Konto. Die Ermächtigung zum Einzug erteile ich hiermit.

Name des Kontoinhabers

Bankleitzahl , Konto-Nr.

Geldinstitut Ort des Geldinstituts

Bankeinzug kann nur innerhalb Deutschlands und nur von einem Giro- oder Postscheckkonto erfolgen.



Bitte im (Fenster-)Briefumschlag einsenden.
Nicht als Postkarte verwenden!

INPUT64

Vertriebsabteilung
Verlag Heinz Heise GmbH
Postfach 61 04 07

3000 Hannover 61