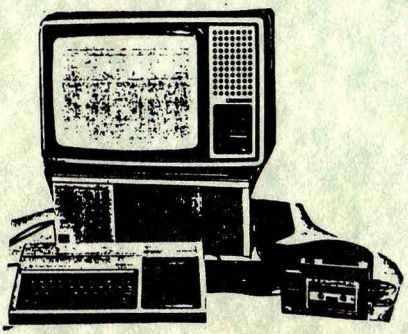
RAUSCH & HAUB

Assemblerkurs III

für Fortgeschrittene

```
 99/4 ASSEMBLER
VERSION 1.2                                          PAGE 0006
  0147 046A 0420          BLWP @VMBW                 *an VDP-RAM uebergeben
       046C 045C
  0148                    *
  0149 046E 0200          LI   R0,>0384              *R0 = Farben im VDP-Ram (Adr.)
       0470 0384
  0150 0472 0201          LI   R1,COLOR              *R1 = Startadresse Chr.-Farben
       0474 03E4
  0151 0476 0202          LI   R2,12                 *12 Bytes (6x2) schreiben
       0478 000C
  0152 047A 0420          BLWP @VMBW                 *an VDP-RAM uebergeben
       047C 046C
  0153                    *
  0154 047E 0200          LI   R0,>0701              *R0 = Bildschirmfarbe
```



Hagera (R)

www.tiggiuc.it

Ti 99/4a

H A G E R A

A S S E M B L E R - K U R S III , 1. AUFLAGE

AUTOR: Hans-Georg Rausch

RAUSCH + HAUB
Vertriebsgesellschaft dbR
Postfach 320313

5300 BONN 3

Hans-Georg Rausch
Assembler Kurs für den TI-99/4a Homecomputer
in Verbindung mit dem Editor/Assembler-Paket
und einer Speichererweiterung.

Voraussetzung ist ferner die Kenntnis der Assemblersprache des TI-99/4a in ihren Grundzügen, wie sie durch ASSEMBLER KURS II, ebenfalls von uns herausgegeben, vermittelt wird.

Eine Diskette gehört nicht zum Lieferumfang dieses Bandes.

1. Fassung 1985

5300 Bonn.

VORWORT

Nach dem großen Interesse an unserem Kurs II für Einsteiger haben wir uns dazu entschlossen, diesen Kurs III herauszubringen. Für den Fall, daß Sie noch nicht mit dem Assembler - insbesondere dem Editor/Assembler-Paket von Texas Instruments - gearbeitet haben, empfehlen wir Ihnen unbedingt, sich zunächst diesen Kurs II als Einsteigerkurs anzuschaffen. Die Arbeit mit dem vorliegendem Buch setzt die Kenntnis des TI-Assemblers in seinen Grundzügen voraus.

Nach diesem Kurs sollten Sie in der Lage sein, eigenständig auch kompliziertere Probleme zu lösen.

Einige Übungsaufgaben sollen Ihnen helfen, die Kapitel besser zu verstehen. Zu den meißten werden wir auch wieder fertige Lösungen anbieten - diesmal allerdings nicht auf Diskette, da ein Großteil der Aufgaben theoretischer Natur ist und keinen sichtbaren Effekt hat.

Von zwei HAGERA(R)-Programmpaketen haben wir in diesem Kurs Ausschnittweise Listings veröffentlicht. Wer an den vollständigen Programmen interessiert ist, kann diese von uns beziehen. Näheres dazu im Anhang.

Hans-Georg Rausch
(Autor)

Zu diesem Kurs

Um mit diesem Kurs tiefer in die Materie der Maschinensprache des TI-99/4a einzusteigen, sollten Sie folgende Konfiguration zur Verfügung haben:

- Konsole TI-99/4a
- Monitor, Fernseher etc.
- Editor/Assembler Paket
- Speichererweiterung
- Diskettenlaufwerk
- Drucker
- Extended Basic Modul

Auf Drucker und Extended Basic kann zur Nor verzichtet werden. Allerdings werden wir in diesem Buch auch insbesondere auf das Ansprechen von Maschinenprogrammen aus dem Basic eingehen, was natürlich besonders für Extended Basic Besitzer von Interesse ist.

Der Kurs ist in mehrere Kapitel unterteilt.

Im ersten Kapitel finden Sie den eigentlichen Kurs mit weiterführenden Informationen zum Assembler des TI-99/4a. Nachdem Sie die Unterprogramme für Bildschirmausgabe etc. aus Kurs II kennen, möchten wir jetzt etwas weiter ausholen und auch Tips für eigene Programme geben.
Sie erfahren etwas über die Funktionsweise der bisher noch nicht behandelten Maschinenbefehle, den Umgang mit Sprites, das Ansprechen verschiedener Arbeitsmodi auf dem TI, den Zugriff auf Maschinenprogramme aus dem Basic und die Übergabe von Parametern sowie noch einiges Nützliche mehr.

Im zweiten Kapitel finden Sie zu einem Großteil der Aufgaben, die Ihre Arbeit in Kapitel 1 unterstützen, Lösungsvorschläge. Die dort gemachten Vorschläge müssen keineswegs die einzige oder beste mögliche Lösung sein. Wir überlassen es dabei Ihnen, besser Wege zur Problemlösung zu finden.

Kapitel 3 beinhaltet im Wesentlichen zwei Listings. Es handelt sich dabei um Auszüge aus MODE CONTROL und TORPEDO BASIC, zwei Programmpakete von HAGERA(R), von denen das erste bereits seit längerer Zeit erfolgreich ist und das zweite gleichzeitig mit diesem Kurs erstmals herausgegeben wird. Die vollständigen Pakete können direkt von uns bezogen werden. Beachten Sie unsere Hinweise im Anhang.

Das vierte Kapitel gibt Ihnen Anwendungsvorschläge zu den in diesem Buch vorgestellten Teilprogrammen, Routinen und Utilities.

Damit Sie einen schnellen Überblick über die Assembler-Sprache gewinnen und die Adressierung des TI-99/4a Ihnen keine Schwierigkeiten bereitet, stellen wir in Kapitel 5 eine Reihe von Tabellen und Listen zur Verfügung. Wesentlich ist dort auch eine Auflistung der Mnemonics, die jedoch keineswegs das Handbuch ersetzen soll oder kann.

Im Anhang finden Sie dann noch die bereits erwähnten Diskettenempfehlungen zu diesem Kurs.

Wir hoffen nun, daß Sie an diesem Band gefallen finden und würden uns über eine Nachricht von Ihnen sehr freuen. Beiliegend erhalten Sie zwei Postkarten: Unsere Service-Karte senden Sie bitte innerhalb von 14 Tagen an uns ab. Nur, wenn wir diese Karte zurückerhalten (auf der Sie uns übrigens auch Ihre Kritik an diesem Band mitteilen können), gelangen Sie in den Genuß der besonderen HAGERA(R)-Serviceleistungen. Näheres dazu finden Sie in unseren Infos und Broschüren.

ANMERKUNG:
Trotz sorgfältiger Überarbeitung kann dieses Buch keinesfalls alle Fragen, die im Zusammenhang mit dem Erlernen der Assembler Sprache auftreten, klären. Auch kann nicht garantiert werden, daß dieses Buch völlig frei von Fehlern ist. Unsere Haftung beschränkt sich daher in jedem Fall auf den Kaufpreis dieses Buches, insbesondere dann, wenn ein Schaden durch unsachgemäße Benutzung, fehlerhafte, falsche oder mangelnde Information entstanden ist. Beschriebene Grundlagen beziehen sich auf die uns bekannten Tatsachen.

INHALT

Kapitel	Inhalt	Seite
	Autorennachweis	6
	Vorwort	7
	Zu diesem Kurs	8
	Inhaltsverzeichnis	10
	Quellennachweis	13
	Urheberrechtlicher Hinweis	13
1.	ASSEMBLER KURS III	17
1. 1.	Logische Instruktionen	19
1. 2.	Bitverschiebungen in Registern	33
1. 3.	Sprites in Maschinensprache	45
1. 4.	Der Grafik-Modus des TI-99/4a	61
1. 5.	Der Text-Modus des TI-99/4a	71
1. 6.	Der Multicolor-Modus des TI-99/4a	87
1. 7.	Der Bit-Map-Modus des TI-99/4a	113
1. 8.	Basic-Zugriff auf Maschinenprogramme	151
1. 9.	Parameterübergabe Basic <--> MC	171
1.10.	MC-Verkettung mit Extended Basic	209
1.11.	Erstellen des Speicherauszugs	219
1.12.	Die CRU-Instruktionen	245

INHALT

Kapitel	Inhalt	Seite
2.	LÖSUNGEN ZU DEN ÜBUNGSAUFGABEN	253
2. 1.	Lösungen zu Kapitel 1.1.	257
2. 2.	Lösungen zu Kapitel 1.2.	263
2. 3.	Lösungen zu Kapitel 1.3.	266
2. 4.	Lösungen zu Kapitel 1.4.	270
2. 5.	Lösungen zu Kapitel 1.5.	273
2. 6.	Lösungen zu Kapitel 1.6.	274
2. 7.	Lösungen zu Kapitel 1.7.	276
2. 8.	Lösungen zu Kapitel 1.8.	285
2. 9.	Lösungen zu Kapitel 1.9.	289
2.10.	Lösungen zu Kapitel 1.10.	301
2.11.	Lösungen zu Kapitel 1.11.	303
2.12.	Lösungen zu Kapitel 1.12.	306
3.	SOFTWARE	309
3. 1.	MODE-UMSCHALTUNG (aus Mode Control)	312
3. 2.	VDPEEK und VDPOKE (aus(Torpedo Basic)	315
4.	ANWENDUNGSHINWEISE	317

INHALT

Kapitel	Inhalt	Seite
5.	TABELLEN UND LISTEN	321
5. 1.	Mnemonic-Übersicht	325
5. 2.	Register und VDP-Register	335
5. 3.	Farbtabelle	336
5. 4.	Extended Basic Gleichstellungen	336
5. 5.	Utility-Routinen in Extended Basic	337
5. 6.	Fehlermeldungs-Verkettung	338
5. 7.	Warnungsmeldung-Verkettung	339
5. 8.	Radix-100-Schreibweise	339
5. 9.	Benutzung der Speichererweiterung XB + E/A	340
5.10.	VDP-RAM Benutzung in verschiedenen Modi	341
	ANHANG	343
	Diskettenempfehlungen	345
	Nachwort	348

Eine Diskette gehört nicht zum Inhalt dieses Bandes.

Die Benutzungsbedingungen werden mit dem Erwerb des Kurses anerkannt.

QUELLENNACHWEIS

Texas Instruments Learning Center
(Handbuch zum Editor Assembler)

HAGERA(R) Assembler Kurs II

HAGERA(R) Mode Control

HAGERA(R) Torpedo Basic Systemerweiterung

URHEBERRECHTLICHER HINWEIS

ASSEMBLER KURS III (C) HAGERA

KRITIKEN ZUM ASSEMBLER KURS II:

COMPUTER KONTAKT, die bekannte Zeitschrift für alle Anwender, schieb in Ausgabe 10/85: Der Kurs führt Anfänger sehr gut in die Materie ein, ohne in die teilweise doch recht trockene Theorie abzuschweifen...

USER-STIMMEN belegen den Erfolg: Sehr Empfehlenswert, Für Anfänger sehr gut geeignet, hervorragend, sehr gut... dies waren nur einige der 'Lobeshymnen'. Nicht zuletzt wurde immer wieder gewünscht, diesen Kurs doch zu erweitern, was wir nun endlich auch getan haben.

Immer wieder wird in Computerzeischriften und Club-Infos positives über den Kurs II berichtet - und das von Anwendern, die es ja schließlich wissen müssen.

Nicht ganz unschuldig an dem positiven Echo ist sicher auch die beiliegende Diskette. Angefangen bei einer einfachen Routine für die Bildschirmausgabe bis hin zum kompletten Spiel ist alles enthalten. Damit ist es möglich, alle erlernten Schritte sofort am Computer nachzuvollziehen. Wir beginnen auch nicht mit grauer Theorie - Ergebnisse sind vielmehr sofort am Bildschirm sichtbar. Zu jedem Kapitel gibt es kleine Übungsaufgaben, die eine Vertiefung des Erlernten Stoffes ohne Probleme ermöglichen, und natürlich sind auch Musterlösungen vorhanden.

LIEFERUMFANG UND PREISE

ASSEMBLER KURS II:

1 gebundenes Buch, 344 Seiten Assemblersprachen-Einführung;
1 Diskette mit Musterlösungen und Spielprogramm im Quellen- und Objekt-Code

DM 79.90

ASSEMBLER KURS III (C) HAGERA

ASSEMBLER KURS II FÜR EINSTEIGER

Umfang:

- 344 Seiten und eine Programmdiskette.

Inhalt:

- Umgang mit dem Editor-Assembler
- Bildschirmausgabe
- Zeichen vom Bildschirm lesen
- Definieren von Sonderzeichen
- Farben Definition
- Abfrage der Tastatur
- Töne programmieren
- Mathematik und Programmtechnik
- Schleifen, Verzweigungen und Sprünge
- Verschieben und Vergleichen
- Arithmetische und logische Befehle
- Problemstellungen
- Software: Spiele-Listing
- Übungsaufgaben und Lösungen
- Assembler-Mnemonics (Befehle)
- Erläuterungen fremder Begriffe
- Anhang mit Liste der Diskettenfiles

ACHTUNG: CLUBS, SAMMELBESTELLER UND WIEDERVERKÄUFER

Für CLUBS gibt es ab sofort ermäßigte CLUB-Preise, wenn mehr als ein Exemplar zur gleichen Zeit an die gleiche Adresse angefordert wird. Verlangen Sie die Aktuelle Sonderpreisliste für Sammelbesteller und Clubs. Gewerbsmäßige Händler fordern bitte unsere Händlerpreisliste an!

1.1

1.1. LOGISCHE OPERATIONEN

Im KURS II haben wir Ihnen gezeigt, wie man mit Hilfe der verschiedenen eingebauten Unterroutinen wie VMBW, VSBW und VWTR ein Zeichen auf den Bildschirm bringt, etwas auf den Bildschirm schreibt oder Farben ändert. Jedes dieser Programme ersetzte für uns eine Reihe von einzelnen Maschinenbefehlen.

Bei vielen Assembler-Versionen sind solche Unterprogramme nicht vorhanden. Dies bedeutet, man muß sich um alles selbst kümmern. Jeder, der ein Maschinensprache-Handbuch in die Finger bekommt, wird zunächst mit Erstaunen feststellen, daß gebräuchliche Routinen, vom Basic her bekannt, einfach fehlen; hier insbesondere PRINT, DISPLAY, INPUT, CALL KEY oder ähnliche.

Dies macht jedoch den Unterschied zwischen Assembler und einer höheren Programmiersprache aus. In Basic führen die meißten Befehle zu einem direkten, sichtbaren Ergebnis. In Assembler hingegen sind eine Reihe von Einzelanweisungen erforderlich, um etwas bestimmtes zu erreichen. Warum?

Jeder Computer arbeitet intern mit zwei unterschiedlichen Zuständen. Diese sind 'Strom fließt' und 'Strom fließt nicht'. Alles andere müssen wir dem Computer erst beibringen - Auch der Basic-Interpreter ist schließlich nicht dem 'Gehirn' des TI-99/4a entsprungen, sondern ein Mensch mußte ihn entwickeln. Bei einem Computer kommt uns nun zugute, daß er einen solchen Zustand speichern kann. Jeden Speicher muß man sich als Schalter vorstellen, der solange seinen Zustand beibehält, bis wir diesen wieder ändern. Einen solchen Speicher/Schalter nennt man Bit.

Mit einem einzelnen Bit lassen sich zwei Zustände ausdrücken. Nehmen wir einmal an, 'Strom fließt' würde die Zahl '1' und 'Strom fließt nicht' die Zahl '0' repräsentieren. Damit wären wir der Arbeitsweise unseres Rechners schon ein gutes Stück näher gekommen. Wenn man nun mehrere Dieser Speicher/Schalter, also mehrere Bits, miteinander verbindet, müssten sich damit doch Zahlen darstellen lassen, die wir vom Basic gewohnt sind. Wichtig ist, daß wir vorher zwei kleine Vereinbarungen treffen:

1. Das am weitesten rechts stehende Bit soll den niedrigsten Wert verkörpern.

2. Jedem Bit wird mit einem Exponent von 2 multipliziert, wobei der Exponent aus der Position des Bits in der Kette berechnet wird. Die äußerste rechte Position sei Position '0'.

Für die folgende Bit-Kette wäre dann eine Wertberechnung möglich: '01100101'.

Der tatsächliche Wert ergibt sich aus:

```
1 x 2 hoch 0 (1x1)                =   1
0 x 2 hoch 1 (0x2)                =   0
1 x 2 hoch 2 (1x2x2)              =   4
0 x 2 hoch 3 (1x2x2x2)            =   0
0 x 2 hoch 4 (0x2x2x2x2)          =   0
1 x 2 hoch 5 (1x2x2x2x2x2)        =  32
1 x 2 hoch 6 (1x2x2x2x2x2x2)      =  64
0 x 2 hoch 7 (0x2x2x2x2x2x2x2)    =   0
      Summe  (1+4+32+64)          = 101
```

Wenn Sie sich bisher gefragt haben, wo denn die Zahlen bleiben, mit denen Sie arbeiten, wissen Sie jetzt, daß der Computer diese Zahlen wie gezeigt ablegt. Man nennt diese Form 'binär', weil das Binäre Zahlensystem (Zustände 0 und 1) im Gegensatz zu dem uns geläufigen Dezimalen Zahlensystem (Zustände 0,1,2,..,8,9) nur genau zwei Zustände darstellen kann.

Komplizierter wird das Ganze, wenn wir mit negativen Zahlen arbeiten möchten. Im binären Zahlensystem gibt es kein 'negatives Vorzeichen'. Wir treffen dafür mit dem Computer eine weitere Vereinbarung:

3. Wenn das an der äußersten linken Stelle befindliche Bit gesetzt ist (Wert 1), so sei der folgende Zahlenwert negativ. Andernfalls sei er positiv. Bei einem negativen Wert ist dieser Bit für Bit umzukehren und anschließend '1' zu addieren.

Setzen wir also in der erläuterten Bitkette das erste Bit auf '1', so ergibt sich ein völlig anderer Wert:

Wert: 11100101.

Umkehrung: 00011010
Addieren: 1
Ergebnis: 00011011

Nach unserem oben beschriebenen System ergibt sich daraus:

x=(1x2 0)+(1x2 1)+(1x2 3)+(1x2 4).

Das Ergebnis lautet: 27.

11100101 ist also die Binärdarstellung von '-27'.

Der Vorteil eines Computers liegt nun darin, daß er große Mengen von Schaltern blitzschnell in ihrem Wert verändern kann, und daß es möglich ist, den Zustand der Schalter weiter zu bearbeiten. So ist es möglich, Schalterzustände auf andere Schalter zu übertragen oder Zustände zu addieren. Jeweils 8 oder 16 Schalter (Bits) können zu einer Einheit zusammengefasst werden. Der TI ist in der Lage, eine solche Einheit gleichzeitig von einer Speicherstelle (Große Ansammlung von Schalter-Einheiten) an eine andere weiterzugeben, zu kopieren oder anders zu verarbeiten.

Derartige Vorgänge können jedoch nicht ohne weiteres auf dem Bildschirm beobachtet werden. Auch lässt sich eine Bitbewegung nicht direkt akustisch wahrnehmbar machen. Es sind jedoch gerade die Maschinenbefehle, welche nicht direkt etwas sicht- oder hörbares bewirken, welche die Rechenzeit eines Computers oder die Wirkungsweise eines Programms ausmachen.

Unsere eingangs erwähnten Unterroutinen beinhalten viele solcher Befehle. Eine Reihe davon haben wir bereits in Kurs II vorgestellt, ohne uns näher mit ihnen zu beschäftigen. Während Ihnen also der Umgang mit Wortbefehlen wie MOV, C, etc. oder Bytebefehlen wie MOVB, CB etc. bekannt sein müsste, haben solche wie ANDI, ORI, SZCB und ähnliche immer noch einen fremden Character. Es handelt sich dabei um logische Befehle und Instruktionen, welche innerhalb des Computers, für den Programmierer nicht direkt sichtbar, zu einem resultat führen.

Damit sind wir an einem Punkt angelangt, den wir im Einsteigerkurs tunlichst vermieden haben: Bei der Theorie. Je langweiliger diese aber auch anmuten mag - ohne Theorie werden Sie kaum in der Lage sein, ein Assembler-Programm zu schreiben. Spätestens bei etwas komplizierteren Routinen, die wir Ihnen mit den Ausschnitten aus MODE CONTROL und TORPEDO BASIC in diesem Buch vorstellen, werden Sie feststellen, daß es unmöglich ist, direkt drauflos zu programmieren, wenn man etwas sinnvolles zustande bringen will.

Eine gute Arbeitsvorbereitung ist von ungeheurer Wichtigkeit. Daher halten Sie sich in diesem wie auch in den folgenden Kapiteln an die abschließenden Problemstellungen, wenn Sie die Funktionsweise der Mnemonics ausprobieren möchten.

Die Instruktionen, mit denen wir uns nachfolgend näher beschäftigen wollen, lauten:

ANDI	AND Immediate	Direkte And-Verknüpfung
ORI	OR Immediate	Direkte Or-Verknüpfung
XOR	Exclusive OR	Nor-Verknüpfung
INV	Invert	Invertieren (umkehren)
CLR	Clear	Gleich zu >0000 setzen
SETO	SEt To One	Gleich zu >FFFF setzen
SOC	Set Ones Corresponding	Gleiche Einsen setzen
SOCB	Set Ones Corresponding, Byte	" bei Bytes
SZC	Set Zeros Corresponding	Gleiche Nullen setzen
SZCB	Set Zeros Corresponding, Byte	" bei Bytes

In einer Übersicht und kurz erläutert finden Sie diese Instruktionen in Kurs II. Eine Ausführliche Beschreibung ist im Assembler-Handbuch, welches zusammen mit dem Editor/Assembler Paket geliefert wird, enthalten.

Wir wollen hier die unterschiedliche Wirkungsweise demonstrieren. Dazu ist es erforderlich, Instruktionen zu unterscheiden, die sich auf ein Byte (8 Bit) beziehen und solche, die sich auf ein Wort (16 Bit) beziehen.

Byte-Operationen sind SOCB und SZCB. Alle anderen Maschinenbefehle arbeiten mit 16 Bit und sind daher Wort-Operationen.

Weiterhin muß unterschieden werden, ob bei einer Instruktion ein Register oder eine Speicherstelle verändert wird und wieviele Operanden erforderlich sind.

Ausschließlich Register werden verändert mit ANDI und ORI (=AND und OR Immediate). Immediate ist Ihnen bereits von AI und CI bekannt. Immediate besagt, daß es sich um die Operation mit einem Register und einer Zahlenkonstanten handelt, wobei diese Konstante das Register direkt und unmittelbar verändert. Schauen wir uns daher an, was geschieht, wenn ein Register mit einem bestimmten Wert geladen ist:

```
Register 1:  20   0000 0000 0001 0100
Konstante :  112  0000 0000 0111 0000
```

```
Befehl:    ANDI R1,112
```

Dieser Befehl ändert den Inhalt des Registers 1 auf 16, da nur jene Bits gesetzt werden, die sowohl im alten Wert von Register 1 als auch in der Zahlenkonstanten gesetzt sind. Dies trifft nur auf Position 4 zu, welche in gesetztem Zustand den Wert 16 verkörpert.

```
Befehl:    ORI  R1,112
```

Dieser Befehl ändert ebenfalls den Inhalt des Registers 1, allerdings auf 116, da alle Bits, die entweder im alten Wert des Registers, in der Konstanten oder in beiden Operanden gesetzt sind, im Register 1 gesetzt werden.

Daraus ergeben sich folgende 'Wahrheitstafeln':

ANDI	0	1
0	0	0
1	0	1

ORI	0	1
0	0	1
1	1	1

XOR verändert ebenfalls ein Register, und zwar auf Grundlage der Nor-Wahrheitstafel. Das heißt, das Registerbit wird gesetzt, wenn das Operandenbit nicht denselben Wert hat.

Beachten Sie die Syntax (Es wird nicht mit einer Konstanten gearbeitet, sondern mit einem Adresseninhalt). Nehmen wir dazu einmal an, unsere 112 stünde in Adresse ADR (>2000). In diesem Falle wären gültige Instruktionen:

```
XOR  §ADR,R1
```

oder

```
XOR  §>2000,R1
```

Das Register ist bei ORI der zweite Operand, der Wert der Verknüpfung der erste. Auch hierzu wieder die Wahrheitstafel:

XOR (Nor)	0	1
0	0	1
1	1	0

(Nor= Not or = XOR = Exclusive Or).

Ähnlich wie XOR arbeiten die Instruktionen SOC, SOCB, SZC und SZCB. Hier ist jedoch kein Register erforderlich - die Bearbeitung kann auch direkt mit zwei Adressen geschehen. Dazu laden wir >4014 einmal in Adresse ADR1 (>2040) und in ADR >2070, und vergleichen wir die unterschiedlichen Auswirkungen der folgenden Befehle:

```
SOC  §ADR,§ADR1 oder SOC §ADR,§>2040 o. SOC §>2000,§ADR1
SOCB §ADR,§ADR1 oder SOCB ...
SZC  §ADR,§ADR1 oder SZC  ...
SZCB §ADR,§ADR1 oder SZCB ...
```

Verändert wird in allen Fällen nur der Zieloperand (der zweite Operand), also der Inhalt von Adresse >2040.

```
SOC §ADR,§ADR1
Inhalt ADR1:        0100 0000 0001 0100
Inhalt ADR:         0010 0000 0111 0000
-----------------------------------------
Ergebnis in ADR1:   0100 0000 0001 0100
```

Im Zieloperanden werden alle Bits gesetzt, die entweder (in unserem Beispiel) in ADR1 oder in ADR und ADR1 gesetzt sind.

```
SOCB §ADR,§ADR1
Inhalt ADR1:        0100 0000 0001 0100
Inhalt ADR:         0010 0000 0111 0000
-----------------------------------------
Ergebnis in ADR1:   0100 0000 0000 0000
```

In den führenden 8 Bit des Zieloperanden werden jene Bits gesetzt, die entweder in ADR 1 oder in ADR und ADR1 Gesetzt sind. Die 8 niedrigwertigen Bits bleiben unverändert.

```
SZC §ADR,§ADR1
Inhalt ADR1:        0100 0000 0001 0100
Inhalt ADR:         0010 0000 0111 0000
-----------------------------------------
Ergebnis in ADR1:   0010 0000 0111 0000
```

Im Zieloperanden werden alle Bits auf Null zurückgesetzt, die entweder nur in ADR1 oder in ADR und ADR1 auf Null gesetzt sind.

```
SZCB §ADR,§ADR1
Inhalt ADR1:        0100 0000 0001 0100
Inhalt ADR:         0010 0000 0111 0000
-----------------------------------------
Ergebnis in ADR1:   0010 0000 0111 0000
```

In den führenden 8 Bit des Zieloperanden werden jene Bits auf Null gesetzt, die entweder nur in ADR1 oder in ADR und ADR1 auf Null gesetzt sind. Die niedrigwertigen Bits bleiben unverändert.

Drei weitere 'Logische Instruktionen' benötigen jeweils nur einen Operanden, da hier etwas bestimmtes mit diesem Operanden geschieht - etwas, das wir nicht weiter beeinflussen können. Diese sind:

INV Operand kehrt die Bitwerte des Operanden um.

CLR Operand setzt alle Bitwerte des Operanden auf Null.

SETO Operand setzt alle Bitwerte des Operanden auf Eins.

Die Vermutung liegt nahe, daß sich diese Instruktionen etwas umständlicher auch mit bereits bekannten nachahmen lassen. Verwenden Sie jedoch möglichst die hier genannten in Ihren Programmen, um Speicherplatz und Rechenzeit zu sparen.

Angenommen, §DUM sei >FFFF.

INV R1 ist das Gleiche wie XOR R1,§DUM

CLR R1 ist das Gleiche wie LI R1,>0000

SETO R1 ist das Gleiche wie LI R1,>FFFF

Eine der Instruktionen, die Sie häufig verwenden, ist mit Sicherheit CLR. Diese wird benutzt, um zum Beispiel Zähler zurückzusetzen oder Flags zu löschen.

Alle Instruktionen können auch mit Adressen benutzt werden:

INV §ADR invertiert den Inhalt von ADR,

CLR §>27C0 löscht die Adressen >27C0 und >27C1

SETO §>A0CA setzt >A0CA auf >FF und >A0CB auf >FF.

Die Operationen sind alle Wort-Operationen, arbeiten also stets mit 16 Bit, wobei das erste der beiden Bytes geradzahlig sein muß, wenn es sich um eine Adresse handelt.

Damit wären wir am Ende unserer Erläuterung zu den logischen Instruktionen angelangt. Die Übungsaufgaben auf der folgenden Seite sollen Ihnen helfen, Anwendungsmöglichkeiten für diese Befehle zu erkennen und Ihre Sicherheit im Umgang mit ihnen verstärken.

ÜBUNGSAUFGABEN

1. Sie haben gesehen, wie der Rechner positive und negative Zahlen als Binärwerte behandelt. Finden Sie jeweils die Binärdarstellung als 16-Bit-Wort von:

9,1056,220,-7,118,513,-512,-1,900,256,-210.

2. Welches Ergebnis erhalten Sie, wenn Sie die folgenden Operanden mit ANDI oder ORI verknüpfen?
Inhalt von R6: >10CB
Festwerte: siehe Aufgabe 1.

3. Welches Ergebnis erhalten Sie, wenn Sie die oben genannten Operanden mit XOR verknüpfen?

4. Mit welcher Instruktion können Sie im führenden Registerbyte die Bits zurücksetzen, die entweder im hochwertigen Byte des Zieloperanden oder im hochwertigen Byte des Quelloperanden eine Null beinhalten, während die niedrigwertigen Bits des Zieloperanden unverändert bleiben?

5. Führen Sie mit den Werten aus Aufgabe 2 die Instruktionen SOC, SOCB, SZC und SZCB aus.

Wenn Sie nicht weiterkommen, lesen Sie noch nicht im Lösungsteil nach, sondern wiederholen Sie Kapitel 1.1! Die genaue Kenntnis dieser Instruktionen ist für das Verständnis der nachfolgenden Abschnitte unbedingt erforderlich!

BROKER - DAS BÖRSENSPIEL

Nach soviel Lernen wird es Zeit, sich ein wenig zu unterhalten. Dazu ist da BÖRSENSPIEL von HAGERA(R) genau das Richtige!

Kaufen und verkaufen Sie Aktien, berufen Sie die Hauptversammlung ein, werden Sie Aufsichtsratsmitglied, hetzen Sie Ihren Gegenspielern den Fiskus auf den Hals!

Der Computer verwaltet alles blitzschnell, wertet Kursveränderungen grafisch aus und verteilt Dividenden.

Ein Spiel für Lange Abende: Jeder Spielstand kann abgespeichert und später wieder eingelesen werden. 2 oder 3 Spieler können gleichzeitig mitspielen - aber natürlich kann auch in Gruppen gespielt werden.

Das BÖRSENSPIEL sollte in Ihrer Sammlung nicht fehlen! Für TI-99/4a, XBasic, 32Kb.

BROKER mit ausführl. Handbuch nur DM 34.90 !!

1.2

1.2. BITVERSCHIEBUNGEN IN REGISTERN

Wie Sie bereits wissen, verfügt der TI-99/4a über eine Reihe von Registern, mit denen verschiedenartigste Berechnungen durchgeführt werden können. Jedes dieser Register ist genau 16 Bit groß.

Manchmal ist es notwendig, innerhalb eines solchen Registers Bits zu verschieben, um etwas ganz bestimmtes zu erreichen. Hierzu stellt uns der Assembler mehrere Instrunktionen zur Verfügung, die sogenannten 'Workspace-Register Verschiebe- Instruktionen'.

Diese lauten:

SRA	Shift Right Arithmetik	Arithmetische Rechtsverschiebung
SRL	Shift Right Logical	Logische Linksverschiebung
SLA	Shift Left Arithmetik	Arithmetische Linksverschiebung
SRC	Shift Right Circular	Rechts-Rotation

In einer Übersicht und kurz erläutert finden Sie diese Instruktionen in Kurs II. Eine ausführliche Beschreibung ist im Assembler-Handbuch, welches zusammen mit dem Editor/Assembler Paket geliefert wird, enthalten.

Auch bei den Verschiebe-Instruktionen möchten wir die unterschiedliche Wirkungsweise demonstrieren. Da alle Operationen 16-Bit-Operationen sind, die nur zusammen mit Registern arbeiten, ist es nicht erforderlich, zu unterscheiden. Die Syntax ist in allen Fällen dieselbe:

<Label> b <Mnemonic> b <Register>,<Zähler> b <Kommentar>

(b steht für ein oder mehrere Leerzeichen, Label und Kommentar sind Optional.)

Wie die Instruktionen im Einzelnen Arbeiten, zeigen die folgenden Beispiele. Vorher müssen wir uns aber mit dem Statusregister beschäftigen, ein Thema, welches wir in Kurs II zwar kurz angeschnitten, aber noch nicht ausführlich behandelt haben.

Das Statusregister enthält Informationen über die jeweils zuletzt durchgeführte Instruktion. Fast jede Instruktion beeinflusst das Statusregister. Ausgenommen davon sind vor allem die Vergleichsinstruktionen und die bedingten Sprünge (beide Instruktionsgruppen testen lediglich bestimmte Bits im Statusregister, verändern diese aber nicht).

Alle Instruktionen, die Bits im Workspaceregister verschieben, können auch das Statusregister beeinflussen.

Die unterschiedliche Bedeutung der Bits im Statusregister wird im Folgenden genau wie im Handbuch zum Editor/Assembler gekennzeichnet.

Bit-Pos.	Kurzbez.	Bedeutung
0	L>	Logisch größer als
1	A>	Arithmetisch größer als
2	EQ	Gleich (Equal)
3	C	Übertrag (Carry)
4	OV	Überlauf (Overflow)
5	OP	Ungerade Parität (Odd Parity)
6	X	Erweiterte Operation (eXtended Operat.)
7-11		Reserviert
12-15		Interrupt Maskierung (Interrupt Mask)

Interessant sind für uns die Positionen 0-4. Nur um diese wollen wir uns kümmern. Eine genaue Auflistung, welches Statusbit durch welche Operation wie beeinflusst wird, finden Sie im Kapitel 5.

Wie Sie noch sehen werden, ist es möglich, durch indirekte Beeinflussung des Statusregisters ganz bestimmte Reaktionen des Computers herbeizuführen. Eine direkte Beeinflussung des Statusregisters, etwa durch bestimmte Befehle, ist nicht möglich.

In der genannten allgemeinen Syntax sehen Sie, daß die Verschiebe-Instruktionen zwei Operanden benötigen: Ein Register und einen Zähler.

Das Register ist selbstverständlich das, in welchem die Verschiebung vorgenommen werden soll.

Der Zähler gibt die Anzahl der Bitstellen an, um die verschoben werden soll. Ist der Wert des Zählers Null, so wird um die Bitanzahl verschoben, die in den vier niedrigwertigsten Bits von Register Null stehen. Ist dann der Inhalt dieser Bits ebenfalls Null, werden 16 Positionen verschoben. Auf diese Weise kann die Zahl der Verschiebungsstellen variabel gehalten werden, obwohl der Zähler eine Konstante ist. Die maximalen Werte für den Zähler sind damit >0 bis >F.

Was bedeutet dies nun für die einzelnen, verschiedenen Instruktionen?

Laden wir dazu einmal bestimmte Werte in unsere Register. Sie wissen doch sicher noch, wie man Register in Programmen lädt? Wenn nicht, sollten Sie sich schnell noch einmal Kurs II ansehen, bevor Sie weiterlesen!

Register 0: 0100 0110 1110 0101
Register 1: 1011 1101 0111 1010

Verschieben wir nun den Inhalt von Register 1 jeweils um eine Bitstelle mit den verschiedenen Instruktionen, und beobachten, was in dem Workspaceregister und was gleichzeitig im Statusregister passiert!

Beachten Sie bei den Beispielen, daß wir immer vom ursprünglichen Wert in Register 1 ausgehen.

SRA R5,1

Register 1: 1011 1101 0111 1010
Verschiebung: 1101 1110 1011 1101

Status : L> wird gesetzt.
A> wird zurückgesetzt.
EQ wird zurückgesetzt.
C wird gesetzt.

Gründe: Der neue Wert von R1 ist logisch (ohne Berücksichtigung des Vorzeichens) größer als der Ursprungswert, aber arithmetisch (mit Berücksichtigung des Vorzeichens) kleiner als der ursprüngliche Wert. Deshalb wird L gesetzt; A> und EQ zurückgesetzt. Das Carrybit enthält anschließend das Vorzeichen des Wertes, wird also gesetzt.

Arithmetisch nach Rechts verschieben heißt, daß links freiwerdende Bitstellen mit dem Wert des Vorzeichenbits gefüllt werden.

SRL R5,1

Register 1: 1011 1101 0111 1010
Verschiebung: 0101 1110 1011 1101

Status : L> wird zurückgesetzt.
A> wird gesetzt.
EQ wird zurückgesetzt.
C wird gesetzt.

Gründe: Der neue Wert von R1 ist logisch (ohne Berücksichtigung des Vorzeichens) kleiner als der Ursprungswert. Deshalb wird L> zurückgesetzt. A> wird jedoch gesetzt, da unter Berücksichtigung des Vorzeichens der neue Wert größer ist als der ursprüngliche. C> enthält anschließend das Vorzeichen des Wertes, wird also gesetzt.

Logisch nach rechts verschieben heißt, daß links freiwerdende Bitstellen mit Nullen aufgefüllt werden.

SLA R5,1

Register 1: 1011 1101 0111 1010
Verschiebung: 0111 1010 1111 0100

Status : L> wird zurückgesetzt.
A> wird gesetzt.
EQ wird zurückgesetzt.
C wird gesetzt.
OV wird gesetzt.

Gründe: Der neue Wert von R1 ist logisch (ohne Berücksichtigung des Vorzeichens) kleiner als der Ursprungswert. Deshalb wird L> zurückgesetzt. A> wird jedoch gesetzt, da unter Berücksichtigung des Vorzeichens der neue Wert größer ist als der ursprüngliche. C> enthält anschließend das Vorzeichen des Wertes, wird also gesetzt. OV wird gesetzt, da sich der Wert des Vorzeichenbits während der Verschiebung ändert.

Arithmetisch nach links verschieben heißt, daß rechts freiwerdende Bitstellen mit Nullen aufgefüllt werden.

SRC R5,1

Register 1: 1011 1101 0111 1010
Verschiebung: 0101 1110 1011 1101

Status : L> wird zurückgesetzt.
A> wird gesetzt.
EQ wird zurückgesetzt.
C wird zurückgesetzt.

Gründe: Der neue Wert von R1 ist logisch (ohne Berücksichtigung des Vorzeichens) kleiner als der Ursprungswert. Deshalb wird L> zurückgesetzt. A> wird jedoch gesetzt, da unter Berücksichtigung des Vorzeichens der neue Wert größer ist als der ursprüngliche. C> enthält anschließend den Wert des Übertragbits.

Zyklisch nach rechts verschieben (Rechts-Rotation) heißt, daß links freiwerdende Bitstellen mit Nullen aufgefüllt werden.

Wie bereits erwähnt, ist es möglich, die Anzahl der Verschiebungen zu variieren, indem man den Zähler auf Null setzt und dadurch um die Anzahl der Bitpositionen verschiebt, die im Workspace-Register Null in den vier niedrigwertigsten Bits untergebracht sind. Schauen wir uns diese 'indirekte' Verschiebung einmal an:

```
SRA R5,0

Register   0: 0100 0110 1110 0101
Register   5: 1011 1101 0111 1010
Verschiebung: 1111 1101 1110 1011

Status      : L> wird gesetzt.
              A> wird zurückgesetzt.
              EQ wird zurückgesetzt.
              C  wird gesetzt.
```

Gründe: Der neue Wert von R1 ist logisch (ohne Berücksichtigung des Vorzeichens) größer als der Ursprungswert, aber arithmetisch (mit Berücksichtigung des Vorzeichens) kleiner als der ursprüngliche Wert. Deshalb wird L gesetzt; A> und EQ zurückgesetzt. Das Carrybit enthält anschließend das Vorzeichen des Wertes, wird also gesetzt.

Ein Beispiel, wie eine derartige Verschiebung im Einzelnen abläuft, finden Sie auch auf Seite 279 in Kurs II (3. überarbeitete Fassung).

Verschoben wird in obigem Beispiel um 5 Position, da die vier niedrigwertigsten Bits in Workspace-Register 0 den Wert 5 beinhalten.

Wie die Verschiebung im Einzelnen abläuft, zeigt die Tabelle auf der folgenden Seite.

```
Register  1: 1011 1101 0111 1010

1. Shift   : 1101 1110 1011 1101

2. Shift   : 1110 1111 0101 1110

3. Shift   : 1111 0111 1010 1111

4. Shift   : 1111 1011 1101 0111

5. Shift   : 1111 1101 1110 1011
```

Nach dem letzten SHIFT befindet sich in Register 1 das Ergebnis der Verschiebung. Anwendungsmöglichkeiten zu diesen Verschiebemöglichkeiten finden Sie unter anderem auch in unseren Ausschnitten zu MODE CONTROL und TORPEDO BASIC in Kapitel 3.

Die Übungsaufgaben auf der nachfolgenden Seite werden sicherlich für Sie kein Problem sein, wenn Sie dieses Kapitel aufmerksam gelesen haben.

Erläuterungen zu den Registern finden Sie in Kapitel 5.

ÜBUNGSAUFGABEN

1. Führen Sie die zuletzt gezeigte indirekte Verschiebung auch mit den Instruktionen SRL, SLA und SRC aus.

2. Welche Instruktion verändert als einzige das Overflow-Bit im Statusregister?

3. Folgende Registerinhalte sollen mit den jeweils angegebenen Instruktionen verschoben werden. Welche Auswirkungen hat dies jeweils auf das Statusregister, und wie lautet der neue Wert des Workspace-Registers 1?

```
R0='225'   R1='>CB0A'   SLA 1,0
R0='0'     R1='>FC10'   SRC 1,4
R0='116'   R1='>0C01'   SLA 1,9
R0='144'   R1='>A008'   SRL 1,0
R0='55'    R1='>3FA5'   SRA 1,5
R0='0'     R1='>1010'   SRC 1,0
R0='0'     R1='>CFA2'   SLA 1,0
```

4. Was ist der Unterschied zwischen einer arithmetischen und einer logischen Instruktion in Bezug auf das Vorzeichen?

5. Das EQ-Statusbit zeigt an, ob der Operand seinen Zustand nicht verändert hat. Ist dies bei den gezeigten Verschiebeoperationen überhaupt möglich, das heißt, kann das EQ-Bit überhaupt gesetzt sein? Nennen Sie, falls ja, ein Beispiel und falls nein, begründen Sie diese Tatsache.

ICE CREAM

Eiskonditor Gianluca Gelatiere muß die Eiswaffeln füllen. Das ist zwischen den Förderbändern der Fabrik aber gar nicht so einfach.

TOLLE GRAFIK, SOUND, SPANNUNG!!!

Dieses Spiel gehört in jede Programmsammlung!

Lassen Sie sich nicht sagen, Sie seien nicht auf dem neuesten Stand. Holen Sie sich jetzt

ICE CREAM

für TI-99/4a, Extended Basic und Joysticks

- auf Diskette oder Cassette nur DM 29,90

1.3

1.3. SPRITES IN MASCHINENSPRACHE

Nachdem wir nun erst einmal genügend Theorie vermittelt haben, wollen wir jetzt wieder zu etwas Praxis übergehen. Wie die Überschrift schon sagt, beschäftigen wir uns mit Sprites.

Wer das Extended Basic Modul besitzt, der weiß die Eigenschaften dieser von der übrigen Bildschirmgrafik unabhängigen Figuren zu schätzen. Bis zu 28 Sprites sind möglich. Jeder von ihnen kann eine andere Farbe haben, sich in eine verschiedene Richtung mit unterschiedlicher Geschwindigkeit bewegen. Wer kein Extended Basic hat, muß auf diese großartige Möglichkeit verzichten - es sei denn, man schreibt sich selbst eine Routine, welche den Zugriff auf Sprites aus dem Basic, nur mit dem Editor/Assembler Paket, erlaubt. Nötig hierzu ist dieses Kapitel, damit Sie wissen, was zu tun ist, um Sprites im Maschinensprache darzustellen.

Wie man diese Maschinenspracheprogramme dann aus dem Basic heraus anspricht, erfahren Sie ebenfalls in diesem Buch. Dazu aber später. Befassen wir uns zunächst mit den Sprites als solchen.

Was wären Spiele ohne rasante, nicht abgehackt wirkende Bewegung auf dem Bildschirm? Langweilig! In Maschinensprache haben Sie nun die Möglichkeit, nicht nur wie in Extended Basic 28, sondern sogar 32 Sprites darzustellen. Selbstverständlich ist auch hier eine Ausdehnungswahl, ähnlich dem MAGNIFY in Extended Basic, möglich.

Um herauszufinden, wie man Sprites in Maschinensprache programmiert, werfen wir wieder einen Blick hinter die Kulissen des TI-99/4a; das heißt, in den Speicher.

Unsere 32 Sprites werden mit den Nummern 0-31, also >0 bis >1F, gekennzeichnet. Erforderlich sind nun weitere Angaben über die Farbe, die Bildschirmposition, den Vergrößerungsfaktor, die Bewegungsrichtung (=Geschwindigkeit) und das Spritemuster. Ein Sprite setzt sich immer aus einem oder aus vier aufeinanderfolgenden Charactern zusammen. Wie Character definiert werden, müssten Sie eigentlich aus Kurs II noch genau wissen.

Der Bildschirm besteht aus 256x192 Pixeln oder Dots. Die Numerierung beginnt in der oberen linken Ecke, und zwar in der ersten Zeile mit >FF, gefolgt von >00, >01, >02 und so weiter bis >BE, was 190 entspricht; in Spaltenrichtung beginnend bei >00 und endend mit >FF. Was es damit auf sich hat, werden Sie noch sehen. Ein Pixel ist bekanntlich der kleinste Punkt, den man auf dem Bildschirm darstellen kann. Je nach Vergrößerungsfaktor kann ein Sprite aus 8x8, 16x16 oder 32x32 Pixelpositionen bestehen. Die obere linke Pixelecke des Sprites ist seine Position.

Die Position eines Sprites wird in der sogenannten SPRITE ATTRIBUTEN LISTE festgelegt. Diese Tabelle enthält, wie Sie noch sehen werden, auch die Farbe des Sprites und einen Hinweis auf das Spritemuster (sein Aussehen).

Jeder Sprite belegt vier Bytes in der Sprite-Attributen-Liste. Zwei davon benötigen wir für die Position. Das erste Byte eines 4-Byte-Blocks definiert wie beschrieben die Y-Position. Wenn ein Sprite sich in der obersten Zeile befindet, enthält das erste Byte >FF, befindet er sich in der zweiten, >00 und so weiter. Das zweite Byte enthält die X-Position, also die Spalte. Diese Bytewerte werden ständig verändert, wenn sich ein Sprite bewegt.

Wenn anstelle der Y-Position, die von >00 bis >BE reicht oder >FF sein kann, ein Wert von >D0 spezifiziert ist, bewirkt dies, daß dieser Sprite und alle folgenden als 'undefiniert' behandelt werden. Wollen Sie hingegen nur einzelne Sprites ausschalten, empfiehlt sich ein anderer Wert größer als >BE und kleiner als >FF.

Die Sprite-Attributen-Liste beginnt im VDP-RAM an Adresse >0300 und reicht bis Adresse >0380. Dies bedeutet, wir können die Positionswerte mit unseren Unterprogrammen VSBW und VMBW festlegen. Bevor wir dies tun, müssen wir uns aber noch um Spritemuster und Farbe kümmern.

Das dritte Byte in jedem 4-Byte-Block verweist auf das Muster, weches den Sprite definiert. Nehmen wir an, das dritte Byte enthält den Wert >41, dann besteht der Sprite aus der Musterdefintion, welche sich an der 41. Position, beginnend bei der Basisadresse der Spritemustertabelle, befindet.

In der Spritemustertabelle werden die Sprites ähnlich abgelegt, wie Sie es von der Musterbeschreibungstabelle der Character (vgl. Kurs II) her kennen.

Jede Musterbeschreibung benötigt 8 Bytes, in denen das Muster pixelzeilenweise definiert ist. Der Code in der Sprite Attributen Liste weist auf die Position; ähnlich, wie es der Zeiger für die Zeichendefinition von Charactern tun sollte, wenn Sie ASCII-Zeichen umdefinieren wollen.

Im Extended Basic ist die Basisadresse der Spritemustertabelle gleich zur Musterbeschreibungstabellen-Startadresse für Character gesetzt, was bewirkt, daß durch Umdefinition von Mustern auch die Spritemuster geändert werden. In Assembler haben Sie jedoch die Möglichkeit, solche Mustercodierungen unabhängig voneinander vorzunehmen.

Die Startadresse wird durch den Inhalt von VDP-Register 6 festgelegt. Der Inhalt dieses Registers, multipliziert mit >800, ergibt die gewünschte Basisadresse.

Durch Setzen dieses Registers auf 0 erreichen Sie denselben Effekt wie im Extended Basic - die Sprites greifen dann auf die ASCII-Muster zu.

Die Farbe des Sprites wird in den vier niedrigwertigsten Bits des vierten Bytes im 4-Byte-Block festgelegt. Damit lassen sich alle 16 möglichen Farben darstellen. Sehen Sie sich dazu unsere Farbtabelle in Kapitel 5 an.

Im vierten Byte des Viererblocks haben wir jetzt die vier hochwertigen Bits übrig. Ihnen kommt eine besondere Bedeutung zu, für deren Erklärung wir wieder auf Extended Basic zurückgreifen möchten.

Bestimmt haben Sie schon beobachtet, daß ein Sprite in Extended Basic nur auf einer Bildschirmseite fließend auf den Bildschirm kommt, während auf der gegenüberliegenden Seite der Sprite auf einmal eingeblendet wird. In Assembler haben Sie die Möglichkeit, die Seite zu wählen, wo der Sprite eingeblendet wird und wo er fließend erscheinen soll. Dies erreichen Sie durch Setzen oder Zurücksetzen des letzten Bits im hochwertigen Nybble, also den hochwertigen vier Bits. Die übrigen Bits sollten auf Null gesetzt sein.

Ein Wert von 1 bewirkt durch Versetzen der internen Spriteposition um 32 Pixelstellen nach rechts, was dazu führt, daß nicht die obere linke, sondern die obere rechte Ecke für das Ein- und Ausblenden ausschlaggebend ist. Ein Wert von 0 führt zum selben Ergebnis wie in Extended Basic.

Damit kennen wir den Aufbau eines Viererblocks in der Sprite-Attributen-Liste:

Byte	Bits	Bedeutung
1	0-7	Y-Position des Sprites
2	0-7	X-Position des Sprites
3	0-7	Hinweiscode auf das Spritemuster
4	0-3	Farbe des Sprites
	4-7	Spriteeinblendungs-Flag

Welche Besonderheiten im Bezug auf Bewegung mit der SPRITE ATTRIBUTEN LISTE beachtet werden müssen, erfahren Sie noch. Kommen wir jetzt zum Vergrößerungsfaktor von Sprites.

Die Sprite-Beschreibungstabelle besteht lediglich aus 2 Bits, und zwar aus den beiden niedrigwertigsten im VDP-Register 1. Ähnlich wie mit dem Magnify-Befehl in Extended Basic können Sie durch Veränderung dieser Bits (Werte von 0-3) die Größe von Sprites und die Anzahl der Character, aus denen sich ein Sprite zusammensetzt, bestimmen.

Dabei bedeuten:

Wert '00': Ein Character in normaler Größe;
Wert '01': Ein Character in doppelter Größe und Breite;
Wert '10': Vier Character in normaler Größe;
Wert '11': Vier Character in doppelter Größe und Breite;

Wenn die Sprite-Beschreibungstabelle vier Character für die Darstellung bestimmt, so werden für das Muster neben dem Code in Byte 3 des 4-Byte-Blocks der Sprite-Attributen-Liste auch die drei folgenden herangezogen. Die Zusammensetzung erfolgt dabei genau wie in Extended Basic. Nehmen wir an, Byte 3 im Viererblock der Sprite-Attributen-Tabelle beinhaltet >44, dann setzt sich ein Sprite, der aus vier Charactern gebildet werden soll, wie folgt zusammen:

```
          :
          :
    >44   :   >46
          :
          :
  --------+--------
          :
          :
    >45   :   >47
          :
          :
```

VDP-Register können Sie mit dem Unterprogramm VWTR ändern. Sollten Sie nicht mehr genau wissen, wie das funktioniert, schlagen Sie noch einmal in Kurs II nach, wo dies ausführlich erläutert wird.

Kommen wir nun zur Sprite-Bewegung.

Vom Basic her ist Ihnen bekannt, daß Sie Sprites, die Sie einmal in Bewegung gesetzt haben, nicht mehr weiter zu kontrollieren brauchen. Die Bewegung obliegt einzig der Kontrolle des Computers. Wie kann aber der Computer entscheiden, ob er einen Sprite bewegen und wann er Ihr Programm weiter ausführen soll?

Das Geheimnis ist der Interrupt, den Sie schon in Kurs II bei der Beschreibung der Sound-Programmierung kennengelernt haben. Um automatische Spritebewegung zu ermöglichen, ist es erforderlich, Interrupts (Unterbrechungen im laufenden Programm, um die Möglichkeit zu geben, etwas ganz anderes zu tun) zu ermöglichen, und zwar spätestens jede 1/60tel Sekunde. Wenn Sie sich erinnern, ermöglichen wir Interrupts mit der Instruktion

LIMI 2

und unterdrücken diese mit der Instruktion

LIMI 0.

Sie sollten sich eine geeignete Stelle suchen, um eine Unterbrechung zu ermöglichen. Nicht jeder Platz im Programm ist dazu geeignet, denn die LIMI 2 Instruktion kann die VDP-Schreibeadresse verändern. Deshalb ist es zum Beispiel nicht ratsam, die Unterbrechung an einer Stelle zu ermöglichen, die selbst Ausgaben an das VDP-Ram macht. Geeignete Stellen sind hingegen:

Warteschleifen;
Tastaturabfragen;
Soundabspielprogramme;
lange Rechenprogramme.

Die Möglichkeit zur Unterbrechung kann direkt aus den aufeinanderfolgenden Befehlen LIMI 2 und LIMI 0 bestehen, oder aus LIMI 2, einer Reihe anderer Befehle (die nicht auf das VDP-RAM zugreifen) und abschließend der Instruktion LIMI 0.

Die Sprite-Bewegungshandhabung ist damit ähnlich der Handhabung von Soundreihen. Beide sind abhängig vom Interrupt, da sie vom Computer kontrolliert werden.

Um die Sprites bewegen zu können, müssen Sie dem Computer außerdem mitteilen, wieviele Sprites Sie überhaupt bewegen wollen.

Diese Anzahl müssen Sie an Adresse >837A setzen.

Wenn Sie beispielsweise die Sprites mit den Nummern 1, 5 und 7 bewegen möchten, so müssen Sie an >837A den Wert 8 setzen, damit sich die Sprites 0,1,2,3,4,5,6 und 7 bewegen können. Die Anzahl muß also immer um 1 höher gesetzt werden als die Spritenummer - bedenken Sie, daß in Maschinensprache die Numerierung mit Sprite '0' beginnt.

Wenn Sie diese Vorbereitungen getroffen haben, können Sie die Richtung und Geschwindigkeit der Sprites durch bestimmte Werte in der SPRITE BEWEGUNGS TABELLE festlegen. Diese Tabelle beginnt immer an Adresse >0780.

Jeder Sprite benötigt 4 Bytes Platz in dieser Tabelle. Das erste Byte legt die Bewegung nach oben und unten (Y-Richtung), das zweite Byte die Bewegung von rechts nach links (X-Richtung) fest. Die jeweils übrigen zwei Bytes des 4-Byte-Blocks werden vom Interrupt-Programm, welches mit der LIMI 2 Instruktion gestartet wird, benötigt. Wir wollen uns jedoch nur um die ersten beiden Bytes kümmern, welche die Geschwindigkeit betreffen.

Vom Extended Basic her kennen Sie die MOTION-Routine. Mit ihr können Gweschwindigkeiten von +127 bis -128 in jede Richtung festgelegt werden, wobei eine negative Zahl die Bewegung nach oben oder links und eine positive Zahl eine Bewegung nach unten oder rechts verursacht.

Die Werte +127 bis -128 können, wie im Kapitel 1 gezeigt, durch hexadezimale Werte von >00 bis >FF dargestellt werden. Daraus ergibt sich, daß alle Werte >00 (0) bis >7F (127) positiv sein müssen und alle Werte von >80 (128; hier aber -127) bis >FF (255, hier aber -1) negativ sein müssen.

Damit haben wir im Prinzip dieselbe Geschwindigkeitsregelung wie im Extended Basic. Der angegebene Wert bestimmt die Anzahl von Bildschirmpixeln, über die sich der Sprite jede 1/60tel Sekunde bewegt. Dies ist auch der Grund dafür, weshalb ein Interrupt spätestens jede 1/60tel Sekunde ermöglicht werden muß.

Fassen wir zusammen:

Um Sprites darzustellen und zu bewegen, müssen Sie

- in der Musterbeschreibungstabelle die ASCII-Character definieren, die den Sprite darstellen sollen;

- in der Sprite-Attributen-Liste die Position, den ASCII-Wert, die Farbe und das Ein/Ausblendungsflag bestimmen;

- in der Sprite-Bewegungstabelle die X- und Y-Richtung und die Geschwindigkeit bestimmen;

- mit LIMI 2 Interrupts zulassen und

- in Adresse >837A die Anzahl der Sprites, die bewegt werden sollen, bestimmen.

Das nachfolgende Programm zeigt die Schritte, die Sie tun müssen, um die Voraussetzungen für die Spritebenutzung zu schaffen.

Ein weiteres Beispiel finden Sie im Handbuch zum Editor/Assembler ab Seite 346.

Die Übungsaufgaben im Anschluß an das Beispielprogramm sollen Ihnen helfen, den Umgang mit Sprites zu beherrschen.

BEISPIELPROGRAMM: SPRITES


```
       IDT  'SPRITE'
*
       DEF  SPRITE
       REF  VSBW,VMBW,VMBR,VWTR
*
SHIP   DATA >3C3C,>3C18,>7E7E,>42A5
*
SPRATR DATA >5080,>800F
       DATA >6472,>8003
       DATA >7264,>8004
       DATA >8058,>800B
       DATA >D000
*
SPRMOV DATA >0400,>0000,>F808,>0000
       DATA >0CFC,>0000,>FCF0,>0000
*
OWNWS  BSS >20
*
*
SPRITE LWPI OWNWS          *Eigene Register laden
       LI   R0,>0384       *R0 mit Farbtabellen-Basisadresse laden
       LI   R1,>1100       *Farbe in R1 kopieren
       BLWP @VSBW          *Farbe in Tabelle eintragen
*
       LI   R0,>0701       *Hintergrundfarbe laden
       BLWP @VWTR          *Ans VDP-Register uebergeben
*
       LI   R0,>0400       *R0 mit Musterbeschreibungs-Basisadresse laden
       LI   R1,SHIP        *Laden des Raumschiffmusters
       LI   R2,>8          *8 Musterbytes zu schreiben
       BLWP @VMBW          *Musterbytes in Tabelle eintragen
*
       LI   R0,>0300       *R0 mit Sprite-Attributen-Basisadresse laden
       LI   R1,SPRATR      *R1 mit Attributen-Pointer laden
       LI   R2,>11         *>11 Bytes zu schreiben
       BLWP @VMBW          *Bytes in Tabelle eintragen
*
```

```
      LI   R0,>07B0     *R0 mit Spritebewegungs-Basisadresse laden
      LI   R1,SPRMOV    *R1 mit Bewegungs-Pointer laden
      LI   R2,>10       *>10 Bytes zu schreiben
      BLWP @VMBW        *Bytes in Tabelle eintragen
*
      LI   R1,>0400     *4 Sprites in Highbyte laden
      MOVB R1,@>837A    *und in Bewegungsmengenspeicher kopieren
*
NEXTSE LI  R0,>300      *Attributen-Start laden
      LI   R5,4         *Counter laden
      LI   R2,2         *2 Bytes zu lesen
*
NEXTIN LIMI 2           *Unterbrechungssequenz
      LIMI 0
*
      LI   R1,OWNWS+>E  *Adresse von R7 in R1 laden
      BLWP @VMBR        *und schreiben
      AI   R7,-24       *24 Pixel subtrahieren
      CI   R7,>B8C8     *Grenze ueberschritten?
      JH   NEU          *Wenn ja, Neubeginn
*
NEXTSP AI  R0,4         *Naechster Sprite
      DEC  R5           *Zaehler vermindern
      JEQ  NEXTSE       *Wenn alle durch, naechste Serie
      JMP  NEXTIN       *sonst naechste Unterbrechung
*
NEU   LI   R1,SPRATR    *Sprite-Attributenpointer laden
      LI   R2,2         *2 Bytes schreiben
      BLWP @VMBW
      JMP  NEXTSP       *Naechster Sprite
*
      END
```

ÜBUNGSAUFGABEN

1. An welchen Stellen im Programm sollte keine Unterbrechung mit LIMI 2 ermöglicht werden und was ist der Grund dafür?

2. In der Sprite-Attributen-Liste werden Ein/Ausblendung, X-Koordinate, Y-Koordinate, ASCII-Muster und Farbe des Sprites bestimmt. Wieviel Speicherplatz benötigt die gesamte Tabelle, wieviel die Information für einen Sprite und in welchen Bytes sind welche Informationen abgelegt?

3. Wie können Sie erreichen, daß ein Sprite auf dem Bildschirm nicht sichtbar ist?

4. Wie können Sie erreichen, daß ein Sprite und alle weiteren mit höherer Nummer auf dem Bildschirm nicht sichtbar sind?

5. Schreiben Sie ein kleines Programm, welches als Sprite einen Lastwagen definiert, der sich von rechts nach links mit einer Geschwindigkeit von 40 Pixeln pro Sekunde bewegt. Der Lastwagen soll aus 4 Charactern bestehen und auf dem Bildschirm 16x16 Pixelpositionen einnehmen. Darüberhinaus soll er plötzlich auf dem rechten Bildschirmrand erscheinen, links aber 'langsam' verschwinden.

6. Lösen Sie jetzt alle Aufgaben noch einmal, ohne im Kapitel 3 oder im Lösungsteil nachzusehen (wie Sie es ja warscheinlich getan haben). Sie wollen doch ASSEMBLER lernen, oder?

SUPER GRAFIK SET TI

In diesem Set ist alles enthalten, was Sie zum entwerfen von Grafikbildern und Sprites brauchen:

1.: Der CHARACTER GENERATOR. Hier werden Figuren auf dem Bildschirm entworfen. Diese können gedreht, gespiegelt, invertiert und abgespeichert werden. Der Computer berechnet auf Wunsch den Hexadezimal-Code, der auch auf einen Drucker ausgegeben werden kann.

2.: Der SPRITE DESIGNER erlaubt das Entwerfen von Sprites jeder Größe, das Testen von Farben und Geschwindigkeiten auf verschiedenem Hintergrund sowie viele Funktionen des Character Generators.

3.: Der R/P-Wandler wandelt abgespeicherte Figuren in Data-Zeilen oder ein fertiges Basic-Unterprogramm um. Dieses können Sie dann später mit MERGE in jedes eigene Programm einbinden.

Alles zusammen auf Diskette für TI-99/4a, Extended Basic und Speichererweiterung.

SUPER GRAFIK SET TI mit Handbuch nur DM 69.90

1.4

1.4. DER GRAFIK-MODUS DES TI-99/4A

Manche wissen es nicht einmal: Der TI-99/4a hat insgesamt vier unterschiedliche Arbeitsmodi, in denen er arbeiten kann. Sie werden alle im Handbuch des TI-99/4a Editor/Assemblers beschrieben - leider aber zu kurz, um sich ein umfassendes Bild von den sich daraus ergebenden Möglichkeiten zu machen. Wir möchten daher in diesem Buch näher auf diese Arbeitsmodi eingehen.

Wenn wir von einem 'Arbeitsmodus' reden, so meinen wir damit, daß der TI-99/4a das VDP-Ram auf eine bestimmte Art und Weise benutzt. In diesem VDP-Ram befinden sich alle wichtigen Tabellen für Bildschirmdarstellung, Musterdefinition, Farbe, Sprite und vieles mehr. Am Interessantesten ist sicher die Möglichkeit, die Bildschirmdarstellung zu verändern.

Bisher kennen Sie lediglich den sogenannten Grafik-Modus. Dies ist der Modus, den Sie vom TI-basic und Extended Basic her gewohnt sind. Er bietet 24x32=768 Bildschirmpositionen, die Sie einzeln ansprechen können, indem Sie ein bestimmtes ASCII-Zeichen auf eine bezeichnete Bildschirmstelle setzen.

Im Kurs II sind wir bereits in allen Einzelheiten auf die Benutzung dieses Modus eingegangen, sodaß wir an dieser Stelle nur Grundlegendes wiederholen möchten.

Zum Arbeiten mit den unterschiedlichen Modi sind die eingebauten Routinen VSBW, VMBW, VSBR, VMBR und VWTR von vorrangiger Bedeutung. Mit ihnen haben Sie direkten Zugriff auf das VDP-Ram.

Um ganz bestimmte Dinge zu ermöglichen, ist das VDP-Ram in Speichersektionen aufgeteilt. Jede Sektion hat eine Aufgabe. Man spricht in Verbindung mit solchen Sektionen auch von Tabellen und Listen, wie sie Ihnen zum Beispiel zuletzt beim Umgang mit Sprites begegnet sind.

Die Tabelle auf der folgenden Seite zeigt diese Einteilung.

VDP-Ram im Grafik-Modus

>0000 - >02FF Bildschirm-Darstellungs-Tabelle

>0300 - >037F Sprite-Attributen-Liste

>0380 - >03FF Tabelle der Characterfarben

>0000 - >0000 freier Platz

>0400 - >077F Sprite-Beschreibungstabelle

>0780 - >07FF Sprite-Bewegungstabelle

>0800 - >09FF Charactermuster-Beschreibungstabelle

>1000 - >37D7 freier Platz

>37D8 - >3FFF Reservierter Block

Das VDP-Ram hat eine Kapazität von >4000 Bytes. Die angegebenen freien und reservierten Plätze haben ebenfalls bestimmte Aufgaben. Soweit erforderlich, werden wir in diesem Buch noch näher darauf eingehen.

Die Adressen des VDP-Ram sind mit denen des freien Benutzerspeichers im unteren Bereich identisch. Das VDP-Ram können Sie daher nur mittels der eingebauten Utilityroutinen, aber nicht direkt ansprechen.

Für die Darstellung des Bildschirms ist nun beispielsweise die Vereinbarung getroffen, daß diese an Adresse >0000 im VDP-Ram beginnt und an Adresse >02FF endet. Dies entspricht genau den 768 Zeichenpositionen, die wir bei einer Darstellung von 24x32 erreichen.

Mit dem Programmteil

```
LI R0,768
LI R1,65
BLWP §VSBW
```

können Sie ein 'A' auf die letzte Bildschirmposition schreiben, während der Programmteil

```
LI R0,769
LI R1,65
BLWP §VSBW
```

eine ganz andere Funktion hätte. Das damit angesprochene Byte gehört nämlich bereits zur Sprite-Attributen-Liste, und ein Wert von 65 an Adresse >769 würde den Sprite Nr. 0 auf die Pixelzeile 65 setzen.

Sie werden in den folgenden Kapiteln sehen, daß diese beiden Unterprogramme völlig andere Bedeutungen haben können - je nachdem, in welchem Modus sie sich befinden. Für jeden Modus ist das VDP-Ram anders eingeteilt.

Anhand der gezeigten Tabelle sehen Sie, daß im VDP-Ram des Grafik-Modus Raum für Sprites, deren Muster und Bewegung vorgesehen ist. Im Grafik Modus sind daher Sprites und Bewegung von Sprites möglich. Dies ist nicht in allen Modi des TI-99/4a so.

Ausschlaggebend dafür, welcher Modus gerade aktiv ist, sind die VDP-Register 0 und 1. Andere VDP-Register regeln die VDP-Speicher Aufteilung und die daraus resultierende Benutzung der eingebauten Unterroutinen für den Zugriff auf diesen Speicherbereich.

Um die verschiedenen Modi des TI-99/4a zugänglich zu machen, müssen folgende Werte in die Register (diese sind bekanntlich 8-Bit-Register) plaziert werden:

Register 0:

Bytes 0-5: Reserviert (müssen auf 0 gesetzt sein)

Byte 6: Modus-Bit 3 (aktiviert Bit-Map-Modus)

Byte 7: Video-Input Ermöglichung/Unterdrückung

Register 1:

Bytes 0-2: Besondere Funktionen

Byte 3: Modus-Bit 1 (aktiviert Text-Modus)

Byte 4: Modus-Bit 2 (aktiviert Multicolor-Modus)

Byte 5: Reserviert (muß auf 0 gesetzt sein)

Byte 6-7: Sprite-Vergrößerungsfaktor

Eine genaue Übersicht über die Funktionen der Bits in den VDP-Registern geben die Tabellen im Kapitel 5.

Wichtig für uns ist also das Bit 3 in VDP-Register 0 und die Bits 3 und 4 im VDP-Register 1. Wenn diese drei Bits alle auf Null gesetzt sind, befinden Sie sich im Grafik-Modus. Dies ist der Ausgangs-Zustand der Register. Wenn Sie sich, wie später noch beschrieben, in einem anderen Mosus als Grafik befinden, können Sie durch Zurücksetzen der genannten Bits in den Registern den alten Zustand (Grafik-Modus) wieder herstellen.

Wenn Sie den Modus ändern möchten, sind einige besondere Vorbereitungen erforderlich, nachdem Sie die Register entsprechend geändert haben.

Möchten Sie in den Bit-Map-Modus umschalten, so ist zunächst nur das Setzen von Bit 6 im VDP-Register 0 erforderlich. VDP-Register können Sie mit der eingebauten Unterroutine VWTR ändern. Genaue Auskunft darüber gibt Kurs II.

Für die Bereitstellung eines anderen Modus muß VDP-Register 1 geändert werden. Dies ist jedoch nicht ohne weiteres möglich. Zwar können Sie eine Änderung mit VWTR herbeiführen; ohne besondere Zusatzinformationen wird diese Änderung jedoch beim nächsten Tastendruck unwirksam. Warum?

Bestimmt haben Sie schon festgestellt, daß in Basic und Extended Basic nach einiger Zeit, in der Sie keine Operation durchgeführt haben, der Bildschirminhalt verschwindet. Sie können den Inhalt durch Betätigung einer Taste wieder sichtbar machen. Dies hat seinen Grund: Wenn eine unveränderte Darstellung längere Zeit auf dem Bildschirm verbleibt, können einzelne Bildpunkte 'ausbrennen'. Dies ist eine Beschädigung der Bildröhre, die zwar nicht allzu schnell vorkommt, trotzdem sollte man lange 'Brennzeiten' vermeiden.

Auch in Assembler ist eine solche 'Bildschirmabschaltung' durchaus möglich. Bestimmt wird dies durch Bit 1 in VDP-Register 1. Ist dieses an, so wird eine Bildschirmabschaltung ermöglicht. Sobald eine Taste gedrückt wird, überträgt der Computer eine Kopie eines ganz bestimmten Speicherinhaltes an VDP-Register 1, und ihr geänderter Wert wird überschrieben. Dies hat den Zweck, daß auf Tastendruck der alte Bildschirminhalt wieder erscheint.

Im Grafik-Modus funktioniert dies ohne weiteres. Wenn Sie jedoch Multicolor oder Text verwenden möchten, müssen Sie den neuen Inhalt, den VDP-Register 1 erhalten soll, auch an jene Speicherstelle übertragen, die bei jedem Tastendruck in VDP-Register 1 übertragen wird.

Diese Adresse ist >83D4.

Eine Änderung von VDP-Register 1 müsste also folgendermaßen aussehen:

```
LI R0,>F001
MOVB R0,@>83D4
SWPB R0
BLWP @VWTR
```

Sie sehen, zuerst wird der neue Wert des Registers zusammen mit der Registernummer ('1') in R0 geladen, und zwar 'verkehrt', damit sich der neue Wert für das VDP-Register im hochwertigen Byte befindet. Dieses hochwertige Byte wird an Adresse >83D4 übertragen. Dann wird der Inhalt von R0 in die richtige Reihenfolge gebracht und mittels VWTR übertragen.

Kommen wir jedoch zunächst zurück zu unserem Grafik-Modus.

Nur in diesem Grafik-Modus arbeitet der TI-Basic-Interpreter. Dies schließt es jedoch nicht aus, die verschiedenen Modi aus einem Basic-Programm heraus anzusprechen. Wir werden dazu in den nächsten Kapiteln einige Tips geben. Einfacher geht es jedoch mit unseren fertigen Programmen, die wir Ihnen am Ende dieses Buches bei den DISKETTENEMPFEHLUNGEN nennen.

Zusammenfassung

Im Grafik-Modus haben Sie die Möglichkeit, einen Character auf eine Bildschirmposition zu bringen. Jeweils ein Charactersatz (Gruppe zu jeweils 8 Charactern) kann in einer anderen Vorder- oder Hintergrundfarbe definiert sein. Darüberhinaus ist die Definition der Bildschirmfarbe und die Darstellung von Sprites möglich. Für Character ist der Bildschirm in 24x32, für Sprites in 192x256 Bildschirmpositionen eingeteilt.

Die Übungsaufgaben auf der folgenden Seite sollen Ihre Kenntnisse über den Grafik-Modus vertiefen. Sofern Sie sich darin sicher fühlen und auch Kurs II gelesen haben, können Sie sofort mit dem nächsten Kapitel beginnen.

ÜBUNGSAUFGABEN

1. Womit wird festgelegt, daß Sie sich im Grafik-Modus befinden?

2. Wo beginnt im Grafik-Modus die Farb-Tabelle und wie können Sie die Farbe der Buchstaben 'H' bis 'O' verändern?

3. Wieviele Bildschirmpositionen hat der Grafik-Modus für die Darstellung von Schrift, Zahlen und Zeichen?

4. Ist im Grafik-Modus möglich:

   - Benutzung von Sprites;
   - Benutzung von automatischer Spritebewegung;
   - Benutzung von verschiedenen Schriftfarben?

5. Welche Größe hat das VDP-RAM?

6. Welches VDP-Register legt die Bildschirmfarbe fest?

7. Warum muß vor einer Veränderung von VDP-Register 1 der neue Wert auch an Adresse >83D4 geschrieben werden?

8. Schreiben Sie ein kurzes Programm, welches im Grafik-Modus das Wort 'ASSEMBLER' in die Bildschirmmitte schreibt.

9. Warum sind die anderen Modi ohne besondere Vorkehrungen aus dem Basic heraus nicht nutzbar?

1.5

1.5. DER TEXT-MODUS DES TI-99/4A

Manch einer von Ihnen hat vielleicht schon die Schneider-, Commodore und Atari-Benutzer beneidet, die 40 oder sogar 80 Zeichen pro Zeile auf dem Bildschirm darstellen können. Für Spiele mögen unsere 32 genügen - aber eine sinnvolle Textbearbeitung oder überhaupt ein Verwaltungsprogramm kommt mit 32 Zeichen nicht aus.

Unser TI-99/4a ist durchaus in der Lage, Zeilen mit 40 Zeichen auf dem Bildschirm darzustellen. Um Sie davon zu überzeugen, bedarf es warscheinlich eines kleinen Beweises!

Laden Sie einmal den EDITOR von Diskette Part A in den Speicher und schauen Sie sich den Edit-Modus genauer an. Schreiben Sie ruhig einmal eine ganze Zeile einfach mit Buchstaben voll, bis der Bildschirm 'umspringt' und den nächsten Ausschnitt sichtbar macht. Kehren Sie dann zum Zeilenanfang zurück (ENTER) und zählen Sie die Buchstaben.

Es wird den einen oder anderen vielleicht überraschen - aber es sind tatsächlich genau 40 Zeichen, die gleichzeitig auf dem Bildschirm in einer Zeile sichtbar sein können. Da sich an den 24 Zeilen nichts ändert, haben wir 24x40=960 Zeichenpositionen zur Verfügung.

Woher nimmt der Computer diesen Platz? An den Bildschirmpixels kann natürlich nichts geändert werden. Dies sind nach wie vor 256 pro Zeile. Es bleibt die Möglichkeit, einen Character schmaler als im Grafik-Modus darzustellen, und genau das wird im Text-Modus getan.

256/32=8 (im Grafik-Modus. denn jeder Character besteht waagerecht (wie auch senkrecht) aus 8 Pixelspalten, also aus 8x8 Pixelpositionen).

256/40=6.4 (im Text-Modus, denn hier besteht zwar jeder Character aus 8 übereinanderliegenden Pixelzeilen, aber aus nur 6 Pixelspalten). Die sechzehn übrigen Pixel auf dem Bildschirm, die sich aus (256 MOD 40) ergeben, bleiben frei.

Sie haben nun gesehen, daß für die Bildschirmdarstellung erheblich mehr Platz benötigt wird, als im Grafik-Modus. Statt 768 benötigen wir 960 Bytes.

Bisher endete die Bildschirmdarstellungs-Tabelle an Adresse >2FF im VDP-Ram. Ein Wert in >300 würde im Grafik-Modus einen Wert in die Sprite-Attributen-Liste schreiben, die dort beginnt. Im Text-Modus müsste dieses Byte jedoch den Inhalt der 769. Bildschirmposition speichern. Betrachten wir uns dazu noch einmal die beiden im letzten Kapitel gezeigten Routinen:

```
LI R0,768
LI R1,65
BLWP §VSBW
```

schreibt im Text-Modus ein 'A' auf die 768. Position des Bildschirms.

```
LI R0,769
LI R1,65
BLWP §VSBW
```

schreibt im Text-Modus ein 'A' auf die 769. Position des Bildschirms.

Dies bedeutet jedoch, daß im Text-Modus an Adresse >0300 keine Sprite-Attributen-Liste beginnen kann.

Im Text-Modus wird auf die Darstellung von Sprites gänzlich verzichtet. Ihre Darstellung ist nicht möglich. Da der Text-Modus jedoch hauptsächlich für Verwaltungsprogramme gedacht ist, hat dies keinerlei Nachteile - im Gegenteil: Dadurch, daß wir auf die Sprite-Tabellen verzichten können, haben wir im VDP-Ram plötzlich jede Menge frei verfügbaren Platz, den wir selbst benutzen können.

Bevor wir jedoch auf die VDP-Speichereinteilung eingehen, noch ein Wort zu den möglichen Farben.

Im Text-Modus gibt es eine Hintergrund- und eine Vordergrundfarbe. Die Farbkombination des Editor/Assemblers ist weiße Schrift auf dunkelblauem Grund. Wie diese Farben festgelegt werden, erfahren sie noch. Wichtig ist, daß wir auch auf die Farb-Tabelle verzichten können.

VDP-RAM im Text-Modus

>0000 - >03BF Bildschirm-Darstellungstabelle

>03C0 - >07FF wird im Text-Modus nicht benötigt

>0800 - >09FF Charactermuster-Beschreibungstabelle

>1000 - >37D7 freier Platz

>37D8 - >3FFF Reservierter Block

Den gewonnenen Platz im VDP-Ram können Sie zum Beispiel nutzen, um einen Bildschirm 'zu verstecken'. Vermutlich wird auf diese Art und Weise beim Editor/Assembler das schnelle 'umblättern' der Seitenausschnitte ermöglicht, die zusammen eine 80-Zeichen-Zeile bereitstellen.

Wie Sie nun den Text-Modus ansprechen können, zeigen die folgenden Schritte.

Um im Text-Modus arbeiten zu können, müssen Sie das entsprechende Modus-Bit im VDP-Register 1 setzen. Wie im letzten Kapitel gezeigt, ist zuvor eine Übertragung des gewünschten neuen Wertes an Adresse >83D4 erforderlich. Mit folgender Sequenz erreichen Sie die Umschaltung:

```
 .
 .
 .
LI R0,>F001
MOVB R0,@>83D4
SWPB R0
BLWP @VWTR
 .
 .
 .
```

Wir haben bereits davon geredet, sind aber noch nicht näher darauf eingegangen: Die Farben im Text-Modus.

Im Text-Modus haben Sie immer genau eine Farbe für die Schrift und eine für den Hintergrund. Die Hintergrundfarbe entspricht der Bildschirmfarbe, und diese wird, wie sie wissen, in VDP-Register 7 festgelegt, und zwar in den niedrigwertigsten 4 Bits.

Wo aber legen wir die Vordergrundfarbe fest, da ja offensichtlich auf eine besondere Farb-Tabelle im VDP-Ram verzichtet wird?

Natürlich ist eine separate Farb-Tabelle nicht erforderlich, da für das Festlegen einer Farbe 4 Bit genügen. Was läge also näher, als die Vordergrundfarbe des Text-Modus ebenfalls in VDP-register 7 unterzubringen, und zwar in den vier hochwertigen Bits?

Der folgende Programmteil setzt die Farbe des Text-Modus auf die Farben, die sie vom Editor/Assembler her gewohnt sind, nämlich weiße Schrift auf dunkelblauem Hintergrund.

```
.
.
.
LI R0,>07F1
BLWP @VSBW
.
.
.
```

Im Grafik-Modus beginnt bei Adresse >300 im VDP-Ram die Sprite-Atributen-Tabelle. Die Werte, die sich dort möglicherweise befinden, werden vom Text-Modus, dessen Bildschirm-Darstellungstabelle ja bis in diesen Teil hineinreicht, zur Darstellung herangezogen. Dies kann unter Umständen sehr phantasievolle Muster auf dem unteren Bildschirmteil ergeben, die aber bestimmt nicht beabsichtigt sind. Deshalb sollten Sie, um derartige ungewollte Darstellungen zu vermeiden, diesen VDP-Bereich löschen - also jedes Byte, welches zur Bildschirmdarstellung herangezogen werden kann, mit Character 32 (>20), dem Leerzeichen, beschreiben. Dies erreichen Sie mit der folgenden Schleife:

```
.
.
.
       LI R0,>0000
       LI R1,>2000
CLEAR  BLWP @VSBW
       INC R0
       CI R0,960
       JLE CLEAR
.
.
.
```

Auf der folgenden Seite finden Sie ein Unterprogramm, welches Sie mit

```
 .
 .
 .
LI R6,>07xx
BL §TXTMOD
 .
 .
 .
```

aufrufen können. Das Programm schaltet in den Text-Modus um und setzt die Farben entsprechend dem niedrigwertigen Byte des DATA-Befehls.

TXTMOD - Umschalten in den Textmodus

```
TXTMOD LI R0,>F001
       MOVB R0,@>83D4
       SWPB R0
       BLWP @VWTR
       MOV R6,R0
       BLWP @VWTR
       LI R0,>0000
       LI R1,>2000
CLEAR  BLWP @VSBW
       INC R0
       CI R0,960
       JLE CLEAR
       RT
```

Im Unterprogramm selbst erfolgt zunächst die Umschaltung in den Text-Modus. Danach wird die Farbe festgelegt, indem das Programm zunächst von Register 6 an Register 0 schiebt und dann an VDP-Register 7 übergibt.

Die Schleife CLEAR löscht alle Bytes, die für die Bildschirmdarstellung herangezogen werden, indem es sie mit Character 32 (>20), dem Leerzeichen, beschreibt.

TXTMOD setzt sich damit im Wesentlichen aus den zwei gezeigten Programmteilen zusammen.

Das Programm auf der folgenden Seite benutzt die Unterroutine TXTMOD für die Umschaltung und bringt eine Zeichenkette auf den Bildschirm. Wenn Sie das Programm gestartet haben, können Sie es nur durch Abschalten des Computers abbrechen.

Schreiben im Text-Modus

```
       DEF START
       REF VMBW,VSBW,VWTR
*
WRITE  TEXT 'DIESE ZEILE HAT GENAU 40 SCHRIFTZEICHEN!'
*
START  LI R6,>07F1
       BL @TXTMOD
       LI R0,>0190
       LI R1,WRITE
       LI R2,40
       BLWP @VMBW
*
LOOP   NOP
       JMP LOOP
*
TXTMOD ...
 .
 .
 .
CLEAR  ...
 .
 .
 .
       RT
*
       END
```

Das Programm START (gleichzeitig die Eingangsstelle, mit der Sie es später starten können), schreibt einen Satz in die Mitte des Bildschirms und demonstriert damit die Benutzung von 40 Zeichen in jeder Zeile. Es setzt die Farben Schwarz/Weiß.

LOOP ist die bereits aus Kurs II bekannte Endlosschleife, welche den Computer bis zum Abschalten 'aktiv' hält, ohne daß etwas passiert. Ohne diese Schleife würde die Programmkontrolle ja in die Unterprogramme TXTMOD/CLEAR geraten, welche Sie dem Hauptprogramm anhängen müssen, damit es lauffähig ist - und dies würde unter Umständen zu einem Systemabsturz führen.

Nachdem Sie jetzt ein Programm kennen, mit dessen Hilfe es sehr einfach ist, in den Text-Modus umzuschalten, möchten wir Ihnen einige Aufgaben stellen, die Ihnen nicht schwerfallen dürften. Allerdings ist Voraussetzung, daß Sie Kurs II - besonders die Benutzung der Tastatur-Abfrage - aufmerksam gelesen haben. Wenn nicht, sollten Sie sich dieses Kapitel vorher noch einmal ansehen.

Die Lösungen zu den folgenden Aufgaben finden Sie teilweise im Lösungsteil dieses Buches, aber auch in den Auszügen aus den HAGERA(R)-Programmen MODE CONTROL II und TORPEDO BASIC, die wir Ihnen in Kapitel 3 vorstellen.

Diese beiden Programme ermöglichen komfortables Arbeiten auch aus dem Basic/Extended Basic heraus und den verschiedenen Arbeitsmodi des TI - insbesondere im Text-modus.

Doch nun zu den Aufgaben.

ÜBUNGSAUFGABEN

1. Schreiben Sie ein kleines Programm, welches es ermöglicht, mittels eines Bildschirmorientiertem Cursors den TEXT-Modus Bildschirm beliebig zu beschreiben (über die Tastatur).

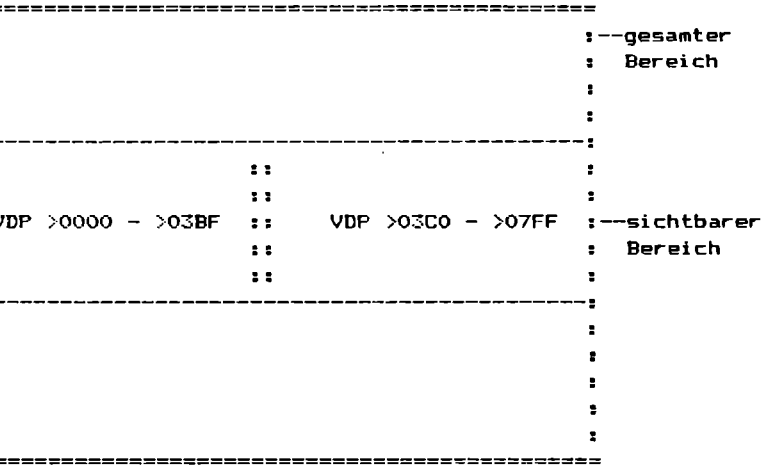
2. Erweitern Sie das erste Programm dahingehend, daß ein Bildschirm in den freien VDP-Bereich kopiert werden kann, so daß zwei komplette Screens für die Eingabe verfügbar sind. Die Umschaltung kann zum Beispiel mit <CTRL>+<TASTE> erfolgen.

3. Erweitern Sie das erste Programm dahingehend, daß wie im Editor-Assembler 80 Zeichen pro Zeile zur Verfügung stehen, indem jeweils einer von drei sich überlappenden Teilen auf dem Bildschirm sichtbar ist. Benutzen Sie auch hier den freien Bereich im VDP-Ram.

4. Erweitern Sie das dritte Programm dahingehend, daß der Bildschirm per Tastendruck gelöscht werden kann. Ferner soll durch Cursor-Up bzw. Cursor-Down der Bildschirm um jeweils eine Zeile nach oben oder unten gescrollt werden; die freiwerdende Zeile wird durch Leerzeichen ersetzt.

5. Erweitern Sie das vierte Programm dahingehend, daß duch Druck einer bestimmten Taste die Bildschirm/Schriftfarbe geändert werden kann; ähnlich, wie Sie es vielleicht vom TI-Writer her kennen (TI-Writer=Textverarbeitungssystem für TI-99/4a).

6. Erweitern Sie das dritte Programm dahingehend, daß im Arbeitsspeicher ein Buffer zur Verfügung gestellt wird. Dieser Buffer soll jeweils weitere Zeilen eines größeren Textes beinhalten, der auf dem Bildschirm nicht sichtbar ist. Der sichtbare Bereich (VDP-Ram) vor dem nicht sichtbaren kann in etwa wie folgt definiert werden:



Durch Cursor-Up und Cursor-Down am oberen bzw. unteren Bildschirmrand soll jeweils eine Zeile aus dem Arbeitsspeicher auf den Bildschirm gebracht werden, während der übrige Bildschirm nach oben/unten scrollt.

7. Zur ordentlichen Textbearbeitung ist es eigentlich nur noch ein kurzer Schritt, wenn man nicht allzu Hohe Ansprüche stellt.

Versuchen Sie doch einmal, ein solches kleines Programm zu entwerfen. Die folgenden Hinweise auf Routinen, die in professionellen Systemen integriert sind, sollen dabei eine Anregung sein.

- Block verschieben

- Block kopieren

- Block löschen

- Absätze einfügen

- Einzelne Zeichen löschen

- Tabulator zum nächsten Wortanfang

- Blättern in unterschiedlichen Schritten

- Text trennen und zusammenfügen

- Zurechtrücken (rechter Randausgleich durch Einfügen von Leerzeichen)

TEXT UTILITIES

Verwaltungsprogramme mit 32 Zeichen sind ein Unding! Deshalb sollten Sie bei solchen Programmen auf den 40-Zeichen-Modus zugreifen, über den wir hier ausführlich schreiben. In diesem Modus werden Verwaltungsprogramme zum reinen Vergnügen.

Sieben Befehle erlauben das Hin- und Herschalten zwischen Text- und Grafikmodus. Damit haben Sie ruck-zuck eine Text- bearbeitung, ein Verwaltungsprogramm oder eine Datei herbeigezaubert.

Mit diesen Utilities machen Sie mehr aus Ihrem TI. Diskette für E/A oder XB mit Speichererweiterung.

TEXT UTILITIES für E/A-Modul-Konf. DM 29.90
TEXT UTILITIES für XB-Modul-Konf. DM 29.90
TEXT UTILITIES für beide Konfig. DM 36.90

1.6

1.6. DER MULTICOLOR-MODUS DES TI-99/4a

Manch einen hat es vielleicht schon einmal gestört, wenn er zur Darstellung einfacher Balkengrafiken erst Sonderzeichen vordefinieren musste, um dann über umständliche und zeitaufwendige Routinen ein Diagramm zu erzeugen. Vielleicht wollten Sie auch nur den Grundriß eines Hauses zeichnen. Für beide Probleme sind weder der Grafikmodus noch (und schon gar nicht) der Text-Modus geeignet.

Damit Sie auf farbige, relativ hoch aufgelöste Bilder trotzdem nicht zu verzichten brauchen, gibt es den Multicolor-Modus. In diesem wird der Bildschirm in 48x64 Positionen unterteilt. Jede Position kann genau eine Farbe enthalten (Bildpunkt gesetzt) - andernfalls besteht die Position aus der Hintergrundfarbe.

Aus dieser Berechnung ergibt sich die Größe von 4x4 Pixeln je Bildschirmposition. Mit dieser relativ hochauflösenden Grafikmöglichkeit lässt sich schon einiges anfangen.

Darüberhinaus lassen sich im Multicolor-Modus Sprites verwenden; ähnlich, wie Sie es vom Grafik-Modus her gewohnt sind.

Für eine solch hochauflösende Grafik muß natürlich das VDP-Ram anders aufgebaut werden, als im Grafik- oder Text-Modus. Sicher kommt dabei der eine oder Andere auf den Gedanken, daß 48x32 Bytes für die Bildschirmbildtabelle zur Verfügung stehen müssen, nämlich genau vier Bit pro Bildschirmposition zur Speicherung des Farbwertes. Im Multicolor-Modus werden schließlich keine Zeichen, sondern Punkte dargestellt. 48x64 Farbspeicherplätze im VDP-Ram würden genau 1536 Nybble = 768 Byte beanspruchen. Dies entspricht der Größe der Bildschirm-Darstellungs-Tabelle des VDP-Ram im Grafikmodus. Man könnte jetzt meinen, daß die Bildschirm-Darstellungstabelle dazu herhalten muß, anstelle der sonst üblichen ASCII-Werte die Farben der Screenpositionen aufzunehmen - aber ganz so einfach ist es denn ja doch nicht.

Um den Multicolor-Modus nutzen zu können, sind einige besondere Vorbereitungen erforderlich. Mit einer Änderung der Benutzungsart des VDP-Ram ist es (leider) nicht getan.

Betrachten wir uns zunächst einmal das VDP-Ram. Tatsächlich belegt die Bildschirm-Darstellungstabelle im Multicolor-Modus genau wie im Grafik-Modus die Adressen >0000 bis >02FF. Diese dienen jedoch keineswegs zur Speicherung der Farben.

Vielmehr muß die Bildschirmdarstellungs-Tabelle vor dem Gebrauch des Multicolor-Modus mit bestimmten ASCII-Werten beschrieben werden.

Wie die Bildschirmdarstellungstabelle initialisiert wird, zeigt die Darstellung auf der folgenden Seite. Zunächst ein paar erläuternde Worte.

In der Tabelle finden Sie am oberen Rand die Spaltennummern und am linken Rand die Zeilennummern; jeweils in dezimaler Schreibweise. Zeile und Spalte ergeben zusammen eine Koordinate, in der Sie den Wert finden, welchen die Bildschirmbildtabelle im Multicolor-Modus enthalten sollte.

Wenn wir davon ausgehen, daß die Bildschirm-Darstellungstabelle an Adresse >0000 im VDP-Ram beginnt und wie im Grafik-Modus von links nach rechts und oben nach unten verläuft, so ergibt sich aus Adresse >0200 die Position in Zeile 16, Spalte 1, wo sich in unserer Tabelle eine >60 befinden soll.

Wie Sie die Werte mit minimalem Aufwand in die Tabelle schreiben können, zeigt das kurze Programm im Anschluß an die Tabelle.

Die Farben, die man zunächst in der Bildschirm-Darstellungs-Tabelle vermutet, findet man in der Muster-Beschreibungstabelle wieder, was eigentlich einleuchtend ist: Im Multicolor-Modus gibt es keine Muster, sondern nur die Möglichkeiten

4x4 Pixel-Quadrat gesetzt oder

4x4 Pixel-Quadrat nicht gesetzt.

Wie Sie nun ein 4x4 Pixel-Quadrat, im Handbuch 'Box' genannt, eine Farbe zuweisen können, dazu werden wir noch kommen. Zunächst einmal zur Initialisierung des Bildschirm-Darstellungs-Tabelle.

BILDSCHIRM-DARSTELLUNGSTABELLE IM GRAFIK-MODUS

	1	2	3	4	5	6	7	8	9	10	11	12	13	.....	29	30	31	32
01:	00:	01:	02:	03:	04:	05:	06:	07:	08:	09:	0A:	0B:	0C:	.....	1C:	1D:	1E:	1F:
02:	00:	01:	02:	03:	04:	05:	06:	07:	08:	09:	0A:	0B:	0C:	.....	1C:	1D:	1E:	1F:
03:	00:	01:	02:	03:	04:	05:	06:	07:	08:	09:	0A:	0B:	0C:	.....	1C:	1D:	1E:	1F:
04:	00:	01:	02:	03:	04:	05:	06:	07:	08:	09:	0A:	0B:	0C:	.....	1C:	1D:	1E:	1F:
05:	20:	21:	22:	23:	24:	25:	26:	27:	28:	29:	2A:	2B:	2C:	.....	3C:	3D:	3E:	3F:
06:	20:	21:	22:	23:	24:	25:	26:	27:	28:	29:	2A:	2B:	2C:	.....	3C:	3D:	3E:	3F:
07:	20:	21:	22:	23:	24:	25:	26:	27:	28:	29:	2A:	2B:	2C:	.....	3C:	3D:	3E:	3F:
08:	20:	21:	22:	23:	24:	25:	26:	27:	28:	29:	2A:	2B:	2C:	.....	3C:	3D:	3E:	3F:
09:	40:	41:	42:	43:	44:	45:	46:	47:	48:	49:	4A:	4B:	4C:	.....	5C:	5D:	5E:	5F:
10:	40:	41:	42:	43:	44:	45:	46:	47:	48:	49:	4A:	4B:	4C:	.....	5C:	5D:	5E:	5F:
11:	40:	41:	42:	43:	44:	45:	4D:	47:	48:	49:	4A:	4B:	4C:	.....	5C:	5D:	5E:	5F:
12:	40:	41:	42:	43:	44:	45:	46:	47:	48:	49:	4A:	4B:	4C:	.....	5C:	5D:	5E:	5F:
13:	60:	61:	62:	63:	64:	65:	66:	67:	68:	69:	6A:	6B:	6C:	.....	7C:	7D:	7E:	7F:
14:	60:	61:	62:	63:	64:	65:	66:	67:	68:	69:	6A:	6B:	6C:	.....	7C:	7D:	7E:	7F:
15:	60:	61:	62:	63:	64:	65:	66:	67:	68:	69:	6A:	6B:	6C:	.....	7C:	7D:	7E:	7F:
16:	60:	61:	62:	63:	64:	65:	66:	67:	68:	69:	6A:	6B:	6C:	.....	7C:	7D:	7E:	7F:
17:	80:	81:	82:	83:	84:	85:	86:	87:	88:	89:	8A:	8B:	8C:	.....	9C:	9D:	9E:	9F:
18:	80:	81:	82:	8D:	8D:	85:	86:	87:	88:	89:	8A:	8B:	8C:	.....	9C:	9D:	9E:	9F:
19:	80:	81:	82:	83:	84:	85:	86:	87:	88:	89:	8A:	8B:	8C:	.....	9C:	9D:	9E:	9F:
20:	80:	81:	82:	83:	84:	85:	86:	87:	88:	89:	8A:	8B:	8C:	.....	9C:	9D:	9E:	9F:
21:	A0:	A1:	A2:	A3:	A4:	A5:	A6:	A7:	A8:	A9:	AA:	AB:	AC:	.....	BC:	BD:	BE:	BF:
22:	A0:	A1:	A2:	A3:	A4:	A5:	A6:	A7:	A8:	A9:	AA:	AB:	AC:	.....	BC:	BD:	BE:	BF:
23:	A0:	A1:	A2:	A3:	A4:	A5:	A6:	A7:	A8:	A9:	AA:	AB:	AC:	.....	BC:	BD:	BE:	BF:
24:	A0:	A1:	A2:	A3:	A4:	A5:	A6:	A7:	A8:	A9:	AA:	AB:	AC:	.....	BC:	BD:	BE:	BF:

MULMOD - Umschalten in den Multicolor-Modus

Um in den Multicolor-Modus umzuschalten, muß natürlich zunächst wieder ein Modus-Bit gesetzt werden. Der Multicolor-Modus wird mit Bit 4 im VDP-Register 1 gesetzt. Beachten Sie, daß eine Kopie des gewünschten neuen Wertes auch hier in Adresse >83D4 übertragen werden muß.

```
      .
      .
      .
MULMOD LI R0,>E101
       MOVB R0,@>83D4
       SWPB R0
       BLWP @VWTR
      .
      .
      .
```

Nun muß die Bildschirm-Darstellungstabelle wie beschrieben mit bestimmten Werten initialisiert werden. Normalerweise beginnt diese an Adresse >0000 im VDP-Ram, wobei wir es in dem nachfolgenden, kurzen Programm auch belassen wollen.

Es ist jedoch möglich, die Startadresse der verschiedenen Tabellen im VDP-Ram zu ändern, und zwar durch andere Werte in bestimmten VDP-Registern, welche Hinweise zur Startadresse der verschiedenen Tabellen enthalten.

Der Hinweis zur Startadresse der Bildschirm-Darstellungs- Tabelle ist in VDP-Register 2 enthalten. Der Wert in diesem Register, multipliziert mit >400, ergibt die Startadresse der Tabelle.

Wenn VDP-Register 2 eine >00 enthält, beginnt auch die Tabelle an Adresse >0000. Würde das Register zum Beispiel eine >02 enthalten, so beginnt die Bildschirm-Darstellungstabelle an Adresse >800 (>02 x >400).

Auf diese Art und Weise ist es möglich, die Tabellen im VDP-Ram zu verschieben. (Sie können ja einmal versuchen, ihr kleines Textbearbeitungsprogramm aus dem Kapitel TEXT-MODUS mit diesen neuen Kenntnissen zu verbessern!!!)

Wir wollen jetzt jedoch zunächst die Bildschirm-Darstellungs-Tabelle initialisieren, wobei wir annehmen, daß diese Tabelle tatsächlich an VDP-Adresse >0000 beginnt.

```
 .
 .
 .
ZEILE BYTE >20
BLOCK BYTE >04
 .
 .
      LI R0,>0000
      LI R3,>0000
      LI R7,>0006
NEXTK MOVB @BLOCK,R5
      AI R3,>2000
NEXTJ MOVB @ZEILE,R6
      MOVB R3,R1
NEXTI BLWP @VSBW
      INC R0
      INC R1
      DEC R5
      JNE NEXTI
      DEC R6
      JNE NEXTJ
      DEC R7
      JNE NEXTK
 .
 .
 .
```

Wie bereits erwähnt, werden die Farben in der Musterbeschreibungs-Tabelle gespeichert. Dabei nimmt die Farbbeschreibung jeder Bildschirmposition im Multicolor-Modus ein halbes Byte ein.

Warscheinlich kommt jeder zunächst darauf, daß die Farben einfach in der Reihenfolge gespeichert werden, wie sie auf dem Bildschirm sichtbar sind - also ähnlich wie die ASCII's in der Bildschirm-Darstellungstabelle. Dem ist jedoch nicht so.

Um Farbpunkte im Multicolor-Modus zu setzen, ist schon eine kleine Berechnung des Speicherplatzes notwendig. Die Farben werden nämlich nicht ganz so einfach abgelegt, wie man es sich vorstellen könnte.

Das erste Byte in der Muster-Beschreibungstabelle liefert die Farbwerte für die ersten beiden 4x4-Pixel-Quadrate (Boxen) in der obersten Bildschirmzeile (Bildschirm=40 Zeilen!). Das nächste Byte beinhaltet die Farbwerte für die beiden darunterliegnden Boxen in der nächsten Zeile. Diese Art der Definition setzt sich bis in die achte Zeile des Multicolor- Bildschirms fort, sodaß sich folgende Speicherbelegung ergibt:

	: 1	: 2	: 3	: 4	: ...	64
1	:1. Byte:				: ...	
2	:2. Byte:				: ...	
3	:3. Byte:				: ...	
4	:4. Byte:				: ...	
5	:5. Byte:				: ...	
6	:6. Byte:				: ...	
7	:7. Byte:				: ...	
8	:8. Byte:				: ...	
9	:		:		: ...	

Unter Berücksichtigung, daß der Bildschirm in diesem Modus 48 Zeilen hat, werden Sie feststellen, daß die Farben in der Muster-Beschreibungstabelle jeweils von aufeinanderfolgenden Bytes definiert werden, welche auf dem Bildschirm jeweils durch den gleichen ASCII-Wert gekennzeichnet sind. In unserem Beispiel werden alle Positionen, die in der Bildschirm-Darstellungs-Tabelle den Wert >00 beinhalten, durch aufeinanderfolgende Bytes in der Muster-Beschreibungstabelle auf ihren Farbwert festgelegt.

Es ist nun naheliegend, daß die folgenden Bytes für die Farbwerte genau jene Bildschirmpositionen bestimmen, die in der Bildschirm-Darstellungstabelle den Wert >01 beinhalten. Die weitere Farbdefinition sieht daher folgendermaßen aus:

	1	2	3	4	...	64
1	1. Byte	9. Byte			...	
2	2. Byte	10. Byte			...	
3	3. Byte	11. Byte			...	
4	4. Byte	12. Byte			...	
5	5. Byte	13. Byte			...	
6	6. Byte	14. Byte			...	
7	7. Byte	15. Byte			...	
8	8. Byte	16. Byte			...	
9					...	

.

.

Diese definition wird Spalte für Spalte fortgeführt, bis jeweils die ersten 8 Zeilen des Multicolor-Bildschirms (was 4 Zeilen des normalen Text/Grafik-Bildschirms entspricht) festgelegt sind. In der Bildschirm-Darstellungstabelle wird die letzte Zeile mit >1F definiert, sodaß die Farbdefinition danach an der ersten Position, die >20 enthält, fortgeführt werden müsste, Dies ist die erste Spalte in der 9. Zeile (oder, auf den normalen Screen übertragen, die erste Spalte in der 5. Zeile).

Die Definition wird also wie folgt fortgeführt:

	1	2	3	4	...	63	64
1	1. Byte	9. Byte			...	249. Byte	
2	2. Byte	10. Byte			...	250. Byte	
3	3. Byte	11. Byte			...	251. Byte	
4	4. Byte	12. Byte			...	252. Byte	
5	5. Byte	13. Byte			...	253 Byte	
6	6. Byte	14. Byte			...	254. Byte	
7	7. Byte	15. Byte			...	255. Byte	
8	8. Byte	16. Byte			...	256. Byte	
9	257. B.	265. B.			...	505. Byte	
10	258. B.	266. B.			...	506. Byte	
11	259. B.	267. B.			...	507. Byte	
12	260. B.	268. B.			...	508. Byte	
13	261. B.	269. B.			...	509. Byte	
14	262. B.	270. B.			...	510. Byte	
15	263. B.	271. B.			...	511. Byte	
16	264. B.	272. B.			...	512. Byte	
17					...		

Für die Festlegung der Farben sind also genau 1536 Bytes Speicherplatz erforderlich; dies entspricht >600 im hexadezimalen Zahlensystem. Das bedeutet, daß wir mit den üblichen >200 Byte Speicherplatz keinesfalls auskommen und die Startadresse für die Musterbeschreibungstabelle verschoben werden muß. Zwar wäre im VDP-Ram ab >800 noch genügend Platz vorhanden - allerdings wird der Bereich ab >1000 von sogenannten PAB's, die für die Benutzung von Peripheren Dateien von Bedeutung sind, genutzt. Wenn wir diesen Raum jetzt für unsere Muster-Beschreibungstabelle heranziehen, könnte dies komplikationen mit SAVE, OLD, OPEN und CLOSE-Befehlen geben; vor allem dann, wenn wir Multicolor aus dem Basic heraus ansprechen möchten.

Wenn Sie auf eine Abspeicherung von Daten und Programmen verzichten können, solange Sie sich im Multicolor Modus befinden, können Sie die Startadresse der Musterbeschreibungs- Tabelle beibehalten.

Anderfalls können Sie diese ändern. Möglich ist dies durch Schreiben eines Wertes in VDP-Register 4. Der Inhalt dieses Registers, multipliziert mit >800, ergibt die Startadresse der Muster-Beschreibungstabelle im VDP-Ram. Im Editor/Assembler ist der Standartwert dieses Registers >01, woraus folgt, daß die Tabelle bei >01 x >800 = >800 beginnt.

Wenn Sie den Multicolor-Modus wählen, befinden sich in der Muster-Beschreibungstabelle die Mustercodes Ihrer ASCII-Character. Dies führt dazu, daß diese, als Farbwerte interpretiert, auf dem Bildschirm erscheinen. Sie sollten daher die gesamte Muster-Beschreibungstabelle auf ein und denselben Farbwert setzen. Das folgende Programm tut dies und löscht damit den Bildschirminhalt; vorausgesetzt, die Muster-Beschreibungs-Tabelle beginnt an Adresse >800 im VDP-Ram.

```
       LI R0,>800
       LI R1,>1100
LOESCH BLWP @VSBW
       INC R0
       CI R0,>A32
       JLE LOESCH
```

Das VDP-Ram im Multicolor-Modus

>0000 - >02FF Bildschirm-Darstellungstabelle

>0300 - >037F Sprite-Attributen-Liste

>0380 - >03FF Im Multicolor-Modus unbenutzt

>0400 - >077F Sprite-Beschreibungstabelle

>0800 - >09FF Muster-Beschreibungs-Tabelle

>1000 - >37D6 freier Raum; wird durch PAB's benutzt

>37D7 - >3FFF Reservierter Speicherbereich

Bedenken Sie, daß bei einer Verschiebung der verschiedenen Tabellen der Platz anderen Tabellen und Listen weggenommen werden kann, was unter Umständen einen Systemabsturz zur Folge haben kann.

Beispiel: Sie benutzen die Sprite-Beschreibungstabelle mit für Ihre Farben und setzen die Startadresse der Muster-Beschreibungstabelle entsprechend herunter. In diesem Fall sollten Sie keine Sprites verwenden, da sonst die Farbwerte bei der Spritedarstellung gleichzeitig als Definition des Spritemusters interpretiert werden kann. Dies führt zu sehr seltsamen Darstellungen auf dem Bildschirm.

Nachdem Sie nun in den Multicolor-Modus umgeschaltet haben, stellt sich die Frage, wie es möglich ist, genau eine bestimmte Box mit einer Farbe auszufüllen. Offensichtlich werden die Farben ja nicht der Reihe nach, sondern nach einem anderen Prinzip in der Muster-Beschreibungstabelle abgelegt.

Um eine Farbe zu setzen, müsste man eine 'Formel' finden, welche aus einfachen Koordinaten die genaue Speicherstelle berechnet, die geändert werden soll. Ein besonderes Hindernis dabei ist sicherlich die Tatsache, daß jeweils vier Bits eines Byte die Codierung einer Box darstellen und wir auf diese vier (jeweils hoch- oder niedrigwertigen) zugreifen müssen, ohne die anderen vier Bits zu verändern.

Mit den Eingangs beschriebenen logischen und den gezeigten Verschiebeinstruktionen dürfte dieses Problem aber zu lösen sein.

Gehen wir einmal davon aus, daß unser Bildschirm in ein Koordinaten-System von 48x24 Boxen unterteilt ist, und wir eine Box ansprechen wollen, indem wir in

- in ein Register die X-Koordinate laden;

- in ein Register die Y-Koordinate laden;

- und in ein Register den gewünschten Farbwert laden.

Dazu müssen wir berechnen, welchen Speicherplatz in der Muster-Beschreibungstabelle eine bestimmte Koordinate ansprechen muß, um den richtigen Farbwert zu ändern - und ob es sich bei den zu ändernden vier Bit um die hochwertigen oder um die niedrigwertigen Bits handelt.

Auf den ersten Blick klingt dies sicher kompliziert. Deshalb wollen wir zur näheren Erläuterung zuerst das erforderliche Programm vorstellen und dann auf die einzelnen notwendigen Schritte eingehen.

RUFBOX / SETCOL - Setzen einer Farbe

```
RUFBOX MOV  R7,R0
       SRA  R0,3
       SLA  R0,5
       MOV  R8,R1
       SRA  R1,1
       A    R1,R0
       SLA  R0,3
       A    R5,R0
       ANDI R7,>7
       A    R7,R0
       CLR  R1
       BLWP §VSBR
*
SETCOL SLA  R6,>C
       LI   R7,>F000
       SRA  R8,1
       JNC  SC1
       SRL  R6,>4
       SRL  R7,>4
SC1    SZCB R7,R1
       SOCB R6,R1
       BLWP §VSBW
       RT
```

Wenn Sie die Routine RUFBOX aufrufen, müssen die Register zuvor wie folgt belegt werden:

Register 6 muß die Farbe enthalten.

Register 7 muß die Y-Koordinate enthalten;

Register 8 muß die X-Koordinate enthalten;

In Register 5 ist zusätzlich die Startadresse der Muster-Beschreibungstabelle des VDP-Ram unterzubringen - normalerweise ist dies >800 - aber es kann ja sein, daß Sie diese verändern möchten, und dann brauchen Sie das gezeigte Unterprogramm nicht zu korrigieren, wenn ein Register diesen Wert als Variable enthält.

Die Werte müssen jeweils rechtsbündig, also in den niedrigwertigen Bits des Registers, untergebracht sein. Ein gültiger Aufruf wäre zum Beispiel:

```
LI R5,>800
LI R6,11
LI R7,15
LI R8,25
BL @RUFBOX
```

Bevor die Farbe gesetzt werden kann, muß zunächst ausgerechnet werden, welche Adresse in der Muster-Beschreibungstabelle unsere bestimmte Box festlegt. Dies geschieht im Abschnitt RUFBOX, der die Koordinaten aus den Registern 7 und 8 in eine Adresse der Muster-Beschreibungs-Tabelle umwandelt.

Folgende Schritte werden ausgeführt:

Zuerst wird die Y-Koordinate in Register 0 übertragen. Diese hat in unserem Beispiel den Wert 15. In Register 0 und 7 sind die Bits daher wie folgt gesetzt, nachdem der erste Befehl ausgeführt wurde:

0000 0000 0000 1111

Wir haben gesehen, daß jeweils für 8 Zeilen die Bytes einer Spalte in der Musterbeschreibungstabelle hintereinander liegen. Deshalb ist es für uns wichtig zu wissen, in welchem 8-Zeilen-Block unsere Koordinate liegt. Wir finden dies heraus, indem wir alle Bits, die einen Wert unter 8 haben, streichen.

SRA R0,3 verschiebt den Inhalt des Registers 0 um drei Bitstellen nach rechts, so daß Register 0 nunmehr folgendes enthält:

0000 0000 0000 0001

Diesen Wert schieben wir nun wieder nach links. Weshalb nicht wieder auf die 8er-Stelle, werden Sie noch sehen.

SLA R0,5 führt zu

0000 0000 0010 0000

und damit enthält Register 0 den Wert 32 oder >20.

Diesen Wert lassen wir zunächst einmal stehen und wenden uns der X-Koordinate zu, die sich im Register 8 befindet. Auch diesen Wert kopieren wir zur weiteren Bearbeitung - diesmal aber in Register 1, denn in R0 befindet sich ja unser bereits teilweise bearbeiteter Y-Wert.

In unserem Beispiel hat X einen Wert von 25 (oder >19). In den Registern 1 und 8 haben wir damit folgende Bitsetzung:

0000 0000 0001 1001

Diesen Wert schieben wir nun um eine Bitsposition nach rechts, denn jeweils zwei Farben gehören ja zu einem Byte, und somit müssen alle Bit-Positionen gestrichen werden, die einen kleineren Wert als 2 haben.

In Register 1 erhalten wir damit

0000 0000 0000 1100

und diesen Wert addieren wir im nächsten Schritt zum Inhalt von Register 0.

A R1,R0 führt zu:

```
  R0 0000 0000 0010 0000
+ R1 0000 0000 0000 1100
------------------------
= R0 0000 0000 0010 1100
```

In Register 0 haben wir jetzt den Wert 44 (>2C). Dies kann jedoch nie und nimmer die Adresse für unsere Farbcodierung sein.

Wir schieben den Inhalt von Register 0 zunächst einmal nach links.

SLA R0,3 führt zu

0000 0001 0110 0000

was einen Wert von 352 (>160) darstellt. Damit kommen wir unserer Sache schon erheblich näher. Sehen Sie sich dazu noch einmal unsere Tabelle an, wobei Sie die Spalten- und Reihenbezeichnungen so abändern, daß mit 0 begonnen und mit 47 bzw. 63 aufgehört wird. Verlängern Sie dann die Tabelle auf einem Blatt Papier dahingehend, daß Sie die Byte-Nummern bis zur (jetzt neu numerierten) Spalte 25, Reihe 15, eintragen können.

Es wird Sie wohl kaum mehr überraschen, daß die von uns im Beispiel gewählte Box tatsächlich durch Byte Nr. 352 definiert wird, was einem Wert von >0160 im Hexadezimalen Zahlensystem entspricht.

Diesen Wert brauchen wir nun nur noch zur Startadresse der Muster-Beschreibungstabelle zu addieren, und wir haben unsere Adresse.

```
Beginn der Muster-beschreibungstabelle: >0800
Aktueller Wert im Register 0          : >0160
-----------------------------------------------
Summe aus diesen Werten               : >0960
```

Wir erreichen dies mit 'A R3,R0'.

Unsere Adresse ist also >0960 im VDP-Ram.

Diese Adresse definiert aber die Farbwerte für die Boxen in zwei Spalten. Um herauszufinden, welche der beiden in einem Byte definierten Farben geändert werden muß, müssen wir auch berücksichtigen, ob unser gesuchter Farbwert sich in einem hochwertigen oder niedrigwertigem Nybble (=4 Bit) befindet.

Wenn wir davon ausgehen, daß unsere Spalten von 0 bis 63 durchnummeriert sind, ergibt sich, daß alle Spalten mit gerader Zahl in einem hochwertigen Nybble (=4-Bit) und alle Spalten mit ungerader Zahl in einem niedrigwertigen Nybble festgelegt sein müssen.

Deshalb muß festgestellt werden, ob es sich bei unserer X-Koordinate um eine gerade oder um eine ungerade Zahl handelt.

Um diese Farbe später ändern zu können, laden wir uns diese in Register 1. Dieses stellt kein Problem dar, denn die Adresse, welche den zu ändernden Farbwert enthält, die kennen wir ja jetzt.

Mit CLR R1 löschen wir den Inhalt von Register 1 und laden mit BLWP §VSBR das Farb-Byte (es enthält, wie bereits mehrfach erwähnt, die Farben für zwei Boxen - man kann es gar nicht oft genug wiederholen).

Die Farbe schieben wir um 12 Bitpositionen nach links, um es später bei der Errechnung des Nybbles einfacher zu haben. Wir brauchen nämlich später dann nur noch zu schieben, wenn es sich um eine gerade Adresse handelt.

Anschließend laden wir noch eine Bitmaske, die uns ebenfalls bei der Berechnung helfen soll, in Register 7. Der alte Inhalt dieses Registers, die Y-Achse, wird nicht mehr benötigt.

Um festzustellen, ob eine X-position gerade oder ungerade ist, müssen wir das Bit mit dem Wert 1, also das niedrigwertigste, testen. Dazu nehmen wir die Hilfe des Status-Registers in Anspruch, welches bei einem SRA-Befehl das Carry-Bit gesetzt hat, wenn aus dem Register eine binäre 1 herausgeschoben wird.

Das bedeutet, daß nach SRA R8,1 das Carry-Bit im Status-Register gesetzt sein muß, wenn es sich um eine ungerade Zahl handelt - und es (auf 0) zurückgesetzt sein muß, wenn es sich um eine gerade Zahl handelt, wobei 'Zahl' unsere X-Koordinate (0..63) ist.

Wenn es sich um eine gerade Zahl handelt, müssen wir den neuen Farbwert in das höchstwertige Nybble eines Registers bringen. Ansonsten kann der alte Farbwert an der Stelle stehenbleiben, an den wir ihn geschoben haben - nämlich an zweithochwertiger Stelle.

Entsprechend dem alten Farbwert verschieben wir gegebenenfalls auch unsere 'Bitmaske'.

```
SRL R6,4
SRL R14,4
```

Diese beiden Schritte sind, wohlgemerkt, nur auszuführen, wenn es sich um eine ungerade Adresse handelt. Wenn es sich um eine gerade Adresse handelt, werden diese beiden Befehle durch die Anweisung JNC SC1 umgangen. Wenn Sie sich noch recht erinnern, bewirkt dieser Befehl einen Sprung, wenn das Carry-Bit des Status-Registers nicht gesetzt ist (vgl. Kurs II).

Die folgenden Schritte sind wieder identisch; egal, ob es sich um eine gerade oder ungerade X-Position der Box handelt.

```
SZCB R7,R1
```

setzt die übereinstimmenden Nullen in dem Teil des Bytes, der nicht verändert werden soll. Dadurch bleibt der neue Farbwert erhalten (Oder-Verknüpfung mit >F (Bin=1111).

```
SOCB R6,R1
```

setzt die übereinstimmenden Einsen in dem Teil des Bytes, der verändert werden soll. Dadurch wird der neue Wert gesetzt (Oder-Verknüpfung mit dem neuen Farbwert.

Diesen Wert tragen wir nun in unsere Muster-Beschreibungs-Tabelle ein:

```
BLWP §VSBW
```

Register 0 enthält ja die im ersten Abschnitt berechnete Adresse.

Anschließend können Sie zum Aufrufprogramm zurückkehren.

SOUND UTILITIES

Alle reden von den Grafikeigenschaften des TI-99/4a.

Daß dieser Computer auch hervorragende Soundmöglichkeiten hat, glaubt keiner! Mit diesem Programm,

SOUND UTILITIES

können Sie

- Sounds testen;

- Lieder Komponieren;

- Musiken speichern;

- diese in DATA-Zeilen umwandeln und

- in jedes Programm einbauen.

Ein LINE/LINE-Editor unterstützt Ihre Arbeit!

Tun Sie etwas für Ihre Ohren! Holen Sie sich die SOUND-UTILITIES ins Haus!!!

Diskette für XBasic/32Kb nur DM 39.90

ÜBUNGSAUFGABEN

1. Welche der folgenden Aussagen sind falsch:

a) Sprite-Darstellung ist im Multicolor-Modus nicht möglich.

b) Der Bildschirm des Multicolor-Modus ist in 48x64 Positionen unterteilt.

c) Jede Position enthält einen einfarbigen Block (Box).

d) Die Farben werden in der Farbtabelle gespeichert.

e) Die Farben werden in der Bildschirmdarstellungs- Tabelle gespeichert.

f) Im Multicolor-Modus ist auch automatische Sprite-Bewegung möglich.

g) Die Bildschirmdarstellung im Multicolor-Modus beginnt immer an Adresse >0000 des VDP-Ram.

h) Um den Multicolor-Modus benutzen zu können, reicht es aus, das Modus-Bit im VDP-Register 1 zu setzen und den neuen Wert dieses Registers in Adresse >83D4 zu speichern.

i) Die Muster-Beschreibungstabelle ist im Multicolor-Modus von großer Bedeutung, obwohl keine Musterdefinitionen benötigt werden.

j) Die Bildschirmdarstellungstabelle wird für den Multicolor-Modus mit bestimmten Werten beschrieben, welche sich während der Arbeit in diesem Modus nicht ändern sollten.

k) Bit 3 im VDP-Register 0 entscheidet darüber, ob Sie sich im Multicolor-Modus befinden oder nicht.

l) Es ist möglich, die Startadresse der Musterbeschreibungs-Tabelle zu ändern.

m) Die Startadresse der Musterbeschreibungstabelle entspricht dem Inhalt von VDP-Register 4, multipliziert mit >800.

n) Die Startadresse der Bildschirmbeschreibungstabelle entspricht dem Inhalt von VDP-Register 4, multipliziert mit >800.

o) Für die Speicherung der Farben des Multicolor-Modus sind genau 768 Bytes erforderlich.

p) Für die Speicherung der Farben des Multicolor-Modus sind genau 1536 Bytes erforderlich.

q) Die Anweisung SRA beeinflusst das Carry-Bit des Statusregisters.

r) JNC ADR verzweigt zu ADR, wenn das überfluss-Bit im Status-Register nicht gesetzt ist.

s) Im Multicolor-Modus können auch ASCII-Zeichen dargestellt werden.

t) Es ist möglich, daß durch Initialisierung des Multicolor Modus die Möglichkeit zum Arbeiten mit Peripheriegeräten eingeschränkt wird, wenn nicht bestimmte Vorkehrungen getroffen werden.

2. Was bewirken die Befehle SZCB und SOCB?

3. Versuchen Sie einmal, das Sprite-Beispiel aus Kapitel 3 vor dem Hintergrund einer Multicolor-Grafik laufen zu lassen (Beispiel mit dem Lastwagen).

4. Wozu ist der Multicolor-Modus nicht geeignet?

TRAFFIC

Hugos Hund ist auf die Straße gelaufen, mitten zwischen den tosenden Verkehr. Sie als echter Tierfreund helfen natürlich sofort, den kleinen Ausreißer zu retten.

Die rüden Autofahrer indessen kümmert dies reichlich wenig: LKWs, PKWs, Dampfwalzen und andere Fahrzeuge trachten Ihnen nach dem Leben. Mit jeder Runde wird der Verkehr immer stärker!

TRAFFIC von HABERA(R) kann auf dem TI-99/4a mit Extended Basic und Joysticks gespielt werden. Auch diesem Spiel ist eine ausführliche deutsche Anleitung beigefügt.
Auf Cassette oder Diskette (bitte angeben!)

TRAFFIC - Das rasante Actiongame DM 24,90 !!

MICRORECHNER

Grundlage für den Umgang mit ASSEMBLER ist die Kenntnis und das Rechnen im hexadezimalen Zahlensystem. Rechenkünstler bringen es auf eine beachtliche Geschwindigkeit beim addieren, subtrahieren, dividieren und multiplizieren hexadezimaler Zahlen und auch bei Kombinationsrechnungen in verschiedenen Zahlensystemen.

Wer sich jedoch solche Kopfarbeit ersparen und dadurch schneller programmieren will, sollte einen Taschenrechner besitzen, der eben dies ermöglicht.

Der SHARP EL-540 ist ein solcher Rechner. Neben dieser Funktion arbeitet er au 15 Klammerebenen, kennt den natürlichen und den dekadischen Logarithmus, Sinusfunktionen, Wurzel- und Potenzberechnungen mit verschiedenen Exponenten, statistische Berechnungen.

Es sind keine Batterien erforderlich, denn der Rechner arbeitet mit SOLARZELLEN und ist daher immer einsatzbereit. Als Energiequelle genügt eine Glühbirne.

Unser Angebot: Dieser Rechner für DM 59.90!

1.7

1.7. DER BIT-MAP-MODUS DES TI-99/4a

Wenn man die Besitzer eines Commodore, Schneider oder Atari Computers hört, ist man geneigt, vor Neid zu erblassen. Können diese doch auf Ihrem Gerät tolle, hochauflösende Grafiken darstellen. Jedes der über 49000 Bildschirmpixel ist einzeln ansprechbar, und diese Auflösung erlaubt gestochen scharfe Bilder auf dem Screen - fast schon wie Fotos.

Der TI bringt es bisher auf 64x48 Bildpunkte, wenn man den Multicolor-Modus berücksichtigt. Das ist sehr schön für grobe Grafiken, aber wenn man Bilder zeigen möchte, erwartet man schon etwas mehr.

Vielleicht haben Sie schon einmal versucht, eine hochauflösende Grafik in Basic nachzuahmen. Dies scheiterte dann meißtens an zwei unterschiedlichen Gründen.

Grund 1: Die Bildschirmdarstellung im Basic-üblichen Grafik Modus erlaubt maximal 2 Farben in jedem 8x8 Pixel großen Bereich. Damit ist zwar eine hochauflösende Grafic mittels des CHAR Unterprogramms generell möglich - die Anzahl der möglichen Farben wird allerdings drastisch eingeschränkt. Dies gilt besonders dann, wenn man bedenkt, daß jeweils 8 ASCII-Zeichen - und nur aus solchen ASCII-Zeichen besteht unsere sogenannte hochauflösende Grafik im Grafik Modus - dieselbe Farbcodierung haben.

Grund 2: Es stehen maximal 256 ASCII-Character (in Assemble) zur Verfügung, die wir als Muster codieren können. Nehmen wir an, jede 8x8 Pixel große Bildschirmposition beinhaltet ein anderes Muster, reicht unser Zeichenvorrat gerade für ein Drittel des Bildschirms. In Basic stehen sogar nur 128 Zeichen zur Verfügung - in Extended Basic sind es gar nur noch 112. Hier fallen schließlich auch die Zeichen 0-30 weg, die als Steuerzeichen dienen.

Wenn Sie trotzdem in den Zeitungen von irgendwelchen Wunderprogrammen lesen, die ohne Speichererweiterung und Extended Basic oder Editor Assembler, Mini Memory oder anderen speziellen peripheren Erweiterungen hochauflösende Grafik herbeizaubern wollen, der sei an dieser Stelle gewarnt: Solche Programme können nur auf der Umdefinition von ASCII-Zeichen beruhen - und dies führt wie gezeigt zu einigen Einschränkungen.

Trotz alledem brauchen Sie sich als TI-99/4a Anwender nicht zu verstecken, besonders nicht vor Rechnern, die gerade wegen ihrer hochauflösenden Grafik für so viel Furore sorgen. Der TI-99/4a hat nämlich durchaus die Möglichkeit, wirkliche und echte hochauflösende Grafik darzustellen, und zwar uneingeschränkt auf dem gesamten Bildschirm und ohne den genannten Begrenzungen durch die Anzahl der verfügbaren ASCII-Character. Auch Farben sind in jeder Pixelspalte für jeden Bildpunkt frei wählbar - lediglich in den Pixelzeilen werden jeweils 8 nebeneinanderliegende Punkte zusammengefasst und dies auch nur, damit der Speicherplatzbedarf nicht ins unermeßliche geht.

Wenn Sie also wirklich hochauflösende Grafik wünschen, sollten Sie sich nicht mit dem Umdefinieren von Ascii-Charactern abgeben. Nutzen Sie vielmehr die hervorragenden Möglichkeiten, die Ihnen Ihr guter alter TI-99/4a liefert.

Sicher ist die Benutzung dieser hochauflösenden Grafik nicht ohne einige Vorkenntnisse möglich. Dieses Kapitel soll Ihnen helfen, diese Kenntnisse zu erlangen, wobei wir es nicht versäumen werden, Ihnen quasi eine Möglichkeit frei Haus zu liefern, mit der Sie den BIT-MAP-MODUS des TI-99/4a problemlos initialisieren und anwenden können.

Die Benutzung dieses Modus fällt bestimmt ein wenig aus dem Rahmen dessen, was Sie bisher in Verbindung mit den verschiedenen Arbeitsmodi des TI erfahren haben. Wenn Sie sich mit der Mnemonic des TMS-9900 noch nicht sicher fühlen, sollten Sie daher die befehlserläuternden Kapitel dieses Buches und von Kurs II nochmals ansehen. Auf Befehle werden wir in diesem Kapitel nicht noch einmal eingehen.

Um den BIT-MAP-MODUS benutzen zu können, ist zunächst im VDP-Register 0 ein bestimmtes Bit zu setzen. Hier sehen Sie bereits die erste Unterscheidung: Für Multicolor und Text war jeweils ein Bit in VDP-Register 1 ausschlaggebend.

Gesetzt werden muß Bit 6 in VDP-Register 0. Wie bei der Beschreibung zum Grafik Modus gesehen, müssen die Bits 0-5 dieses Registers je eine 0 enthalten, während Bit 7 gesetzt sein sollte.

Da wir Register 0, und nicht Register 1 ändern, ist es natürlich nicht erforderlich, etwas nach Adresse >83D4 zu kopieren. Der Bit-Map-Modus wird daher mit folgender Sequenz ausgewählt:

```
 .
 .
 .
LI R0,>0003
BLWP @VWTR
 .
 .
 .
```

Um in diesem Modus arbeiten zu können, sind jedoch noch einige weitere Vorbereitungen erforderlich.

Wie bereits erwähnt, sind 256 ASCII-Character verfügbar. Dies würde genau ein Drittel der benötigten Muster für einen Bildschirm beschreiben. Der Bildschirm des Bit-Map-Modus sollte daher dreimal die ASCII-Character >00 bis >FF (0-255) enthalten, so daß der gesamte Bildschirm in 3 Blocks unterteilt ist, von denen jeder von links nach rechts und oben nach unten durchnumeriert ist, was folgendermaßen aussieht:

BILDSCHIRM-Darstellungstabelle im Bit-Map-Modus

	1	2	3	4	5	6	7	8	9	...	30	31	32
1	>00	>01	>02	>03	>04	>05	>06	>07	>08	...	>1D	>1E	>1F
2	>20	>21	>22	>23	>24	>25	>26	>27	>28	...	>3D	>3E	>3F
3	>40	>41	>42	>43	>44	>45	>46	>47	>48	...	>5D	>5E	>5F
4	>60	>61	>62	>63	>64	>65	>66	>67	>68	...	>7D	>7E	>7F
5	>80	>81	>82	>83	>84	>85	>86	>87	>88	...	>9D	>9E	>9F
6	>A0	>A1	>A2	>A3	>A4	>A5	>A6	>A7	>A8	...	>BD	>BE	>BF
7	>C0	>C1	>C2	>C3	>C4	>C5	>C6	>C7	>C8	...	>CD	>CE	>CF
8	>E0	>E1	>E2	>E3	>E4	>E5	>E6	>E7	>E8	...	>FD	>FE	>FF
9	>00	>01	>02	>03	>04	>05	>06	>07	>08	...	>1D	>1E	>1F
10	>20	>21	>22	>23	>24	>25	>26	>27	>28	...	>3D	>3E	>3F
11	>40	>41	>42	>43	>44	>45	>46	>47	>48	...	>5D	>5E	>5F
12	>60	>61	>62	>63	>64	>65	>66	>67	>68	...	>7D	>7E	>7F
13	>80	>81	>82	>83	>84	>85	>86	>87	>88	...	>9D	>9E	>9F
14	>A0	>A1	>A2	>A3	>A4	>A5	>A6	>A7	>A8	...	>BD	>BE	>BF
15	>C0	>C1	>C2	>C3	>C4	>C5	>C6	>C7	>C8	...	>CD	>CE	>CF
16	>E0	>E1	>E2	>E3	>E4	>E5	>E6	>E7	>E8	...	>FD	>FE	>FF
17	>00	>01	>02	>03	>04	>05	>06	>07	>08	...	>1D	>1E	>1F
18	>20	>21	>22	>23	>24	>25	>26	>27	>28	...	>3D	>3E	>3F
19	>40	>41	>42	>43	>44	>45	>46	>47	>48	...	>5D	>5E	>5F
20	>60	>61	>62	>63	>64	>65	>66	>67	>68	...	>7D	>7E	>7F
21	>80	>81	>82	>83	>84	>85	>86	>87	>88	...	>9D	>9E	>9F
22	>A0	>A1	>A2	>A3	>A4	>A5	>A6	>A7	>A8	...	>BD	>BE	>BF
23	>C0	>C1	>C2	>C3	>C4	>C5	>C6	>C7	>C8	...	>CD	>CE	>CF
24	>E0	>E1	>E2	>E3	>E4	>E5	>E6	>E7	>E8	...	>FD	>FE	>FF

Die ASCII-Werte in der Tabelle haben wir Hexadezimal ausgedrückt. Folgendes kleine Programm initialisiert die Bildschirm-Darstellungstabelle auf die gezeigte Art und Weise:

```
.
.
.
       CLR R0
NEXTJ  CLR R1
NEXTI  SWPB R1
       BLWP §VSBW
       SWPB R1
       INC R1
       INC R0
       CI R1,256
       JNE NEXT I
       CI R0,768
       JNE NEXTJ
.
.
.
```

Zunächst werden die Register 0 und 1 gelöscht. In R0 steht damit die erste Adresse der Bildschirm-Darstellungstabelle; im niedrigwertigen Byte des Registers 1 der ASCII-Code des zu schreibenden Zeichens.

Der ASCII-Wert wird ins hochwertige Byte gebracht und dann das Zeichen mit VSBW geschrieben. Danach wird der ASCII-Wert wieder ins niedrigwertige Byte gebracht, um mittels INC diesen Wert einfach zu erhöhen. Auch die Bildschirmadresse muß um 1 erhöht werden.

Solange noch nicht >FF geschrieben wurde, wiederholt sich der Vorgang des Schreibens und Heraufzählens des ASCII-Wertes in Register 1. Sobald >FF aber geschrieben wurde, springt das Programm weiter zurück und setzt R1 wieder auf >00.

Diese Vorgänge wiederholen sich solange, bis die letzte Bildschirmposition beschrieben wurde.

Die Initialisierung ist beendet, wenn R0 >300 enthält, also 768.

Da wir mehr Platz für die spätere Mustercodierung benötigen (768 Zeichen anstelle von 256), ist es ratsam, die Bildschirm-Darstellungstabelle zu verschieben. Im Handbuch wird empfohlen, diese Tabelle an Adresse >1800 im VDP-Ram beginnen zu lassen. Wir halten dies für sinnvoll, da Sie dadurch auch nicht in Schwierigkeiten mit der für PABs reservierten Zone kommen, die bei Adresse >1000 beginnt (es bleibt genügend Raum für PABs vorhanden).

Sie können die Startadresse der Bildschrimdarstellungstabelle ändern, indem Sie einen neuen Wert in VDP-Register 2 schreiben. Der Inhalt dieses Registers, multipliziert mit >400, ergibt bekanntlich die Startadresse. Um diese auf >1800 zu setzen, muß folgende Sequenz in Ihrem Initialisierungsprogramm enthalten sein:

```
 .
 .
 .
LI R0,>0206
BLWP @VWTR
 .
 .
 .
```

>6 * >400 = >1800.

Es ist natürlich auch möglich, eine andere Adresse zu benutzen, wir halten es aber nicht für ratsam.

Wenn Sie die Startadresse der Bildschirmdarstellungstabelle ändern, müssen Sie in Ihrem Initialisierungsprogramm anstelle von CLR R0 folgendes schreiben:

LI R0,neue Startadresse - also zum Beispiel:
LI R0,>1800.

In der Musterbeschreibungstabelle wird nun die gesamte Bildschirmdarstellung untergebracht. Dies ist möglich, weil sie im Bit-Map-Modus anstelle von 256 genau 768 Muster beschreiben kann, und dies entspricht der Anzahl der Character auf dem Bildschirm. Die Muster sind darin mit je 8 Byte genau so definiert, wie die dazugehörenden Character auf dem Bildschirm zu finden sind.

Die Musterbeschreibungstabelle muß im Bit Map Modus entweder an Adresse >0000 oder >2000 starten. Dies erreichen Sie durch Veränderung des VDP-Registers 4. Der Inhalt dieses Registers, multipliziert mit >800, ergibt die Startadresse der Muster-Beschreibungstabelle. Zur Änderung nehmen Sie eine der folgenden Sequenzen in Ihr Initialisierungsprogramm auf.

Für Placierung an Adresse >0000:

```
 .
 .
 .
LI R0,>0400
BLWP @VWTR
 .
 .
 .
```

Für Placierung an Adresse >2000:

```
 .
 .
 .
LI R0,>0404
BLWP @VWTR
```

MODE CONTROL II

Kennen Sie schon die 4 Arbeitsmodi des TI-99/4a?

- Den Grafik-Modus bestimmt, denn er ist aus dem Basic heraus ansprechbar. Dort haben Sie 24x32 Zeichenpositionen zur Verfügung.

Aber es gibt auch

- den Text-Modus mit 24x40 Zeichen, ideal für Verwaltungsprogramme aller Art;

- den Multicolor-Modus mit 48x64 Zeichenpositionen, z.B. für Diagramme, Grundrißzeichnungen etc.;

- den Bitmap-Modus, in dem alle 192x256 Bildschirmpixel einzeln ansprechbar sind.

Dies alles ist natürlich nur in Assembler möglich. 18 neue Maschinenbefehle, voll aus dem Basic/Extended Basic ansprechbar. Wahlweise für Extended Basic oder Editor Assembler Modul lieferbar.

NUR DM 39.90

Wenn Sie später einen Bildpunkt setzen möchten, so können Sie dies durch Änderung in der erweiterten Musterbeschreibungs-Tabelle tun. Diese Tabelle ist, genau wie die Bildschirm-Darstellungstabelle, in drei Sektionen unterteilt. Wenn also das Muster des Zeichens >BA in der zweiten Sektion des Bildschirms durch Setzen eines Punktes verändert wird, ist es erforderlich, das Muster im dazugehörenden Musterbeschreibungsbyte der Musterbeschreibungstabelle zu ändern - das Byte befindet sich dort ebenfalls in der zweiten Sektion.

Ein Programm, mit dem Sie einen bestimmten Punkt setzen können, werden wir Ihnen noch zeigen.

Wir haben bereits erwähnt, daß im Bit-Map-Modus neben der hochauflösenden Grafik auch mehr Farbdefinitionen gegeben werden können. Jeder ASCII-Character kann aus bis zu 16 Farben bestehen, und das ist mehr, als unsere bisher bekannte Farbtabelle schafft.

Aus diesem Grunde ist es nötig, für die Farbtabelle mehr Speicherplatz zu reservieren. Wir setzen daher die Startadresse unserer Farbtabelle an VDP-Adresse >0000 oder >2000; abhängig davon, wo sich unsere Musterbeschreibungstabelle befindet.

Startet diese an >0000, muß unsere Farbtabelle an >2000 starten, und umgekehrt.

In der Farbtabelle werden die Farben der insgesamt 768 verschiedenen Character wie folgt beschrieben.

Ein Charactermuster enthält die gesetzten Punkte, und die ungesetzten Punkte. Die gesetzten Punkte in der Muster-Beschreibungstabelle stellen den Vordergrund, die dort nicht gesetzten Punkte den Hintergrund dar. In der Farbtabelle legen wir nun für jede Pixelspalte, die in der Musterbeschreibungs-Tabelle definiert ist, die Vorder- und die Hintergrundfarbe fest, welche dann für die jeweils gesetzten oder nicht gesetzten Pixel ausschlaggebend ist. Daraus ergibt sich für die Farbtabelle dieselbe Länge wie für die Musterbeschreibungs- Tabelle.

Ein ausführliches Beispiel dazu, wie Muster und Farben im Bit-Map Modus korrespondieren, finden Sie im Handbuch zum Editor/Assembler auf Seite 337.

Um die Startadresse der Farbtabelle zu ändern, müssen Sie einen bestimmten Wert in VDP-Register 3 schreiben. Der Inhalt dieses Registers, multipliziert mit >40, ergibt die gewünschte Startadresse.

Für das Festlegen der Farbtabelle auf >0000 oder >2000 müssen Sie eine der folgenden Sequenzen in Ihr Initialisierungsprogramm aufnehmen:

Für Placierung an Adresse >0000:

```
  .
  .
  .
LI R0,>0300
BLWP @VWTR
  .
  .
  .
```

Für Placierung an Adresse >2000:

```
  .
  .
  .
LI R0,>0380
BLWP @VWTR
  .
  .
  .
```

Im Bit-Map-Modus sind Sprites möglich, jedoch keine automatische Bewegung derselben, da der Platz dieser Tabelle anderweitig benötigt wird. Daraus ergibt sich nun folgender allgemeiner Aufbau für das VDP-Ram im Bit-Map-Modus:

Das VDP-RAM im Bit-Map-Modus

>0000 - >17FF Musterbeschreibungstabelle oder Farbtabelle

>1800 - >1AFF Bildschirm-Darstellungstabelle

>1B00 - >1FFF freier Raum, z.B. für Sprites nutzbar

>2000 - >37FF Farbtabelle oder Musterbeschreibungstabelle

>3800 - >3FFF Reservierter Block

Den freien Raum können Sie zum Beispiel für die Sprite-Attributen-Liste und die Spritemustertabelle benutzen, aber es sind auch andere Möglichkeiten denkbar. Es empfiehlt sich dazu, die Bildschirm-Darstellungstabelle nicht an >1800, sondern höher in den bisher freien Raum zu schieben, so daß die Adresse >1800 für die Spritemustertabelle verfügbar ist.

Die Sprite-Mustertabelle wird durch Setzen eines anderen Wertes in VDP-Register 6 bestimmt. Der Inhalt dieses Registers, multipliziert mit >800, ergibt die neue Startadresse. Die Sprite-Attributen-Liste wird durch einen neuen Wert in VDP-Register 5 bestimmt. Die Neue Adresse ergibt sich aus dem Inhalt dieses Registers, multipliziert mit >80.

Um diese beiden Sprite-Listen in den freien Raum zwischen den anderen zu setzen, empfiehlt es sich, die folgende Sequenz in Ihr Programm aufzunehmen:

```
  .
  .
  .
LI R0,>0207
BLWP @VWTR
LI R0,>0603
BLWP @VWTR
LI R0,>056C
BLWP @VWTR
  .
  .
  .
```

Zuerst wird die Bildschirm-Darstellungstabelle an Adresse >1C00 verschoben. Danach setzen wir die Sprite-Beschreibungstabelle an Adresse >1800 und die Sprite-Attributen-Liste an Adresse >1B00.

Damit haben wir unser VDP-Ram auf alle denkbaren Möglichkeiten vorbereitet.

Bevor Sie mit den nächsten Absätzen beginnen, in denen Sie erfahren, wie man einen Bildpunkt mit einer ganz bestimmten Farbe setzt, sollten Sie sicher sein, die Initialisierung des Bit-Map-Modus zu beherrschen.

Aus diesem Grunde bringen wir in diesem Kapitel bereits an dieser Stelle einige Übungsaufgaben. Nur wenn Sie diese ohne nachzublättern lösen können, sollten Sie in diesem Kurs weiterarbeiten und ansonsten diesen ersten Abschnitt wiederholen.

ÜBUNGSAUFGABEN

1. Warum stößt man beim Versuch, in Extended Basic oder Basic hochauflösende Grafik darzustellen, schnell an die Grenzen des Systems?

2. Welches Bit in welchem VDP-Register initialisiert den Bit-Map-Modus?

3. Wie muß die Bildschirmdarstellungstabelle initialisiert werden, um im Bit-Map-Modus arbeiten zu können?

4. An welcher Adresse des VDP-Ram sollte die Bildschirm-Darstellungstabelle beginnen, wenn Sie mit Sprites arbeiten, und wo, wenn Sie ohne Sprites arbeiten?

5. Wie können Sie die Startadresse der Bildschirm-Darstellungstabelle verändern?

6. Wieviele Bytes benötigt im Bit-Map-Modus die Musterbeschreibungstabelle; wieviele Bytes die Farbtabelle?

7. An welcher Adresse können diese Tabellen beginnen, und wie berechnet sich die Startadresse (wie legt man sie fest)?

8. In Basic/Extended Basic gilt ein Wert der Farbtabelle jeweils für 8 aufeinanderfolgende Character eines Charactersatzes. Wofür gilt ein Farbwert in der Farbtabelle des Bit-Map-Modus?

9. Wie wird es ermöglicht, Pixelgrafik darzustellen, ohne daß mehr Character erforderlich sind?

10. Wie korrespondieren Muster und Farben im Bit-Map-Modus miteinander?

11. Wie bestimmen Sie, an welcher Adresse im VDP-Ram die Spritemustertabelle beginnt?

12. Wozu dient VDP-Register 5?

13. Was bewirkt folgende Anweisung in Ihrem Programm:

```
LI   R0,>056C
BLWP @VWTR
```

14. Welche der folgenden Dinge sind im Bit-Map-Modus nicht oder nicht ohne besondere zusätzliche Programmteile nicht möglich?

a) Darstellung von Sprites

b) Automatische Spritebewegung

c) Ausgabe von Zahlen, Zeichen und Buchstaben

d) Start der Bildschirmdarstellungstabelle an Adresse >0000 im VDP-Ram

e) Start der Farbtabelle an Adresse >0000 im VDP-Ram

f) Änderung der gesamten Bildschirmfarbe

15. Wie können Sie den Bildschirm im Bit-Map-Modus löschen?

ZEICHNEN UND LöSCHEN im Bit-Map-Modus

Sicher waren Sie von der letzten Frage überrascht. Dieses Thema haben wir nämlich noch gar nicht behandelt. Wenn Sie sich trotzdem mit der Frage auseinandergesetzt haben, umso besser. Dann sind Sie nämlich auf dem besten Wege dazu, ein guter Programmierer zu werden.

Vielleicht haben Sie sogar eine Lösung gefunden. Wenn Sie von den richtigen Überlegungen ausgehen, führte Ihr Nachdenken auch sicher zum Erfolg.

Natürlich gibt es im Bit-Map-Modus nicht so etwas wie CALL CLEAR im Basic. Auch eine weithin bekannte Löschroutine würde nicht zum Erfolg führen:

```
.
.
.
       LI R0,>1800
       LI R1,32
LOESCH BLWP @VSBW
       CI R0,>1800
       JNE LOESCH
.
.
.
```

Diese Routine würde die Bildschirmdarstellungstabelle löschen - im Grafik- und Textmodus durchaus beabsichtigt. Im Bit-Map-Modus enthält diese Tabelle aber die wichtige Standartinitialisierung, die auf keinen fall gelöscht werden darf.

Um unseren Bildschirm 'sauber' zu bekommen, müssen wir also anders vorgehen.

Das, was eigentlich auf dem Bildschirm dargestellt wird, nämlich die gesetzten und nicht gesetzten Punkte, wird in der Musterbeschreibungstabelle codiert. Nicht gesetzte Punkte sollen dabei als 'vom Bildschirm gelöscht', gesetzte als 'auf dem Bildschirm sichtbare Punkte' gelten. Damit ist klar, daß die Mustercodierungen aller Character in der Musterbeschreibungs-Tabelle den Wert >00 in allen Bytes enthalten muß. Nur dann sind alle dargestellten Character auf dem Bildschirm in keinem Pixel sichtbar - und dies ist auch der einzige Weg, die Punkte zu löschen:

```
.
.
.
      LI R0,>0000
      LI R1,>0000
MCLEAR BLWP §VSBW
      INC R0
      CI R0,>1800
      JNE MCLEAR
.
.
.
```

Daß diese Routine tatsächlich die gesamte Bildschirmdarstellung löscht, ist davon abhängig, welche Farben in der Farbtabelle gesetzt sind. Wenn die Hintergrundfarbe aller Character nicht gleich der Bildschirmfarbe oder Transparent ist, erscheint jeder einzelne Character und jede Pixelzeile dieses Characters in der für ihn definierten Farbe der nicht sichtbaren Punkte, was zu sehr seltsamen Effekten auf dem Bildschirm führen kann.

Um diese Effekte zu vermeiden, löschen wir auch alle Bytes der Fabtabelle, indem wir diese auf >00 setzen (Vorder- und Hintergrundfarbe = Transparent).

Unsere oben gezeigte Routine müsste folgendermaßen fortgesetzt werden:

```
.
.
.
      LI R0,>2000
      LI R1,>0000
FCLEAR BLWP @VSBW
      INC R0
      CI R0,>3800
      JNE FCLEAR
.
.
.
```

Diese beiden Teilroutinen können Sie auch dann verwenden, wenn Sie mit der Farbtabelle an VDP-Adresse >0000 und mit der Musterbeschreibungstabelle an >2000 beginnen. Die Routinen sind bis auf die Übernahme und Kontrolle dieser Adressen identisch und würden so jeweils die Aufgabe der anderen Routine übernehmen.

Was aber, wenn man nicht den gesamten Bildschirm löschen, sondern nur ein einzelnes Pixel löschen oder auch setzen will?

In diesem Fall muß eine Formel gefunden werden, mit der sich feststellen lässt, wo im VDP-Ram sich die jeweiligen Informationen für ein bestimmtes Pixel befinden - und zwar sowohl die Farb- als auch die Musterinformationen. Damit es am Anfang nicht gar so schwierig wird, wollen wir uns noch nicht um die Farben kümmern. Wir setzen daher mit folgender Routine den Hintergrund aller Character auf Schwarz und den Vordergrund auf grün (Die Farbtabelle beginnt bei >2000):

```
.
.
.
      LI R0,>2000
      LI R1,>C100
SETCOL BLWP @VSBW
      INC R0
      CI R0,>3800
      JNE SETCOL
.
.
.
```

Damit erscheinen die Linien, die wir Zeichnen, in Grün. Alles andere wird, wenn zuvor unsere Musterlöschroutine durchlaufen wird, auf einen schwarzen Hintergrund gesetzt. Sie können den Hintergrund auch durch Bestimmung eines Wertes in VDP-Register 7 setzen und den Hintergrund der Character Transparent wählen, was den gleichen Effekt hat.

Um nun bestimmen zu können, wo ein Pixel des Bildschirminhalts in der Muster-Beschreibungstabelle definiert wird, müssen wir eine kleine Berechnung anstellen. Dazu haben wir eine Tabelle aufgestellt, die einen Überblick über die Bytebedeutung der Muster-Beschreibungstabelle gibt.

Diese Tabelle können Sie später übrigens ohne weiteres auch auf die Farbtabelle anwenden. Aber soweit sind wir noch nicht. Zunächst zu unserer Tabelle.

Im Bit-Map-Modus wird der Bildschirm in 256 Spalten und 192 Zeilen unterteilt. Es wäre nun sinnvoll, eine Umrechnungsart zu finden, die es ermöglicht, diese Koordinaten in eine Adresse umzuwandeln. Um diese Arbeit zu erleichtern, numerieren wir die Pixelzeilen und -spalten von 0-255 bzw. von 0-191.

	0 1 2 3 4 5 6 7	8-15	16-23 ... 240-247	248-255
0	1. Byte	9. B.	17. B. ... 241. B.	249. B.
1	2. Byte	10. B.	18. B. ... 242. B.	250. B.
2	3. Byte	11. B.	19. B. ... 243. B.	251. B.
3	4. Byte	12. B.	20. B. ... 244. B.	252. B.
4	5. Byte	13. B.	21. B. ... 245. B.	253. B.
5	6. Byte	14. B.	22. B. ... 246. B.	254. B.
6	7. Byte	15. B.	23. B. ... 247. B.	255. B.
7	8. Byte	16. B.	24. B. ... 248. B.	256. B.

.
.
.

Hier sehen Sie, in welchen Bytes die Musterdarstellung der ersten Bildschirmzeile untergebracht ist. Die Codierung erfolgt von links nach rechts und von oben nach unten Characterweise. Der erste Character, >00, wird durch die ersten 8 Bytes codiert. Der zweite Character, >01, wird durch die nächsten 8 Byte (9-15) codiert. Diese Codierung setzt sich in der "normalen" Zeile fort bis Character >1F, welcher der letzte Character in der ersten Zeile ist.

Danach wird die Codierung mit Character >20 in der 8. Zeile des Bit-Map-Bildschirms, welches die 2. Zeile des normalen Bildschirms (also die 2. Characterzeile) ist, fortgesetzt. Die Tabelle auf der folgenden Seite zeigt dies.

.
.
.

	0 1 2 3 4 5 6 7	8-15	16-23 ... 175-183	184-191
8	257. Byte	265.	273. ... 497. B.	505. B.
9	258. Byte	266.	274. ... 498. B.	506. B.
10	259. Byte	267.	275. ... 499. B.	507. B.
11	260. Byte	268.	276. ... 500. B.	508. B.
12	261. Byte	269.	277. ... 501. B.	509. B.
13	262. Byte	270.	278. ... 502. B.	510. B.
14	263. Byte	271.	279. ... 503. B.	511. B.
15	264. Byte	272.	280. ... 504. B	512. B.

.
.
.

Diese Tabelle können Sie Characterzeile für Characterzeile fortsetzen. Jede Characterzeile benötigt genau 256 Bytes zur Definition, woraus sich die Gesamtlänge der Tabelle von >1800 (=6144) Bytes ergibt.

Die letzte Characterzeile und damit die letzten 8 Pixelzeilen der Musterbeschreibungstabelle (und der Farbtabelle) werden daher von den Bytes bestimmt, die in der nachfolgenden Tabelle angegeben sind.

Diese Art der Definition muß jetzt noch in ein Programm verwandelt werden, in dem durch einfache Angabe von Koordinaten in bestimmten Registern ein Punkt gesetzt werden kann.

.
.
.

: 0 1 2 3 4 5 6 7	8-15	16-23 ... 175-183	184-191
184 : 5889. Byte	5897	5905 ... 6129.B.	6137.B.
185 : 5890. Byte	5898	5906 ... 6130.B.	6138.B.
186 : 5891. Byte	5899	5907 ... 6131.B.	6139.B.
187 : 5892. Byte	5900	5908 ... 6132.B.	6140.B.
188 : 5893. Byte	5901	5909 ... 6133.B.	6141.B.
189 : 5894. Byte	5902	5910 ... 6134.B.	6142.B.
190 : 5895. Byte	5903	5911 ... 6135.B.	6143 B.
191 : 5896. Byte	5904	5912 ... 6136.B	6144 B.

Nehmen wir einmal an, auf dem Bildschirm soll in Zeile 97, Spalte 35 ein Punkt gesetzt werden. In diesem Fall müsste das Muster des Characters geändert werden, in dessen 8x8 Feld dieser Punkt enthalten ist. Dies lässt sich errechnen:

97/8=12 (Rest unbedeutend)

35/8=4 (Rest ebenfalls unbedeutend)

12=384 (Errechnete Zeilenzahl*Character je Zeile)

384+4=388 (Errechneter Character+Spaltenposition)

Einen Character '388' gibt es jedoch nicht. Deshalb muß solange 256 abgezogen werden, bis eine Zahl unter 256 herauskommt. In diesem Beispiel erreichen wir dies nach der ersten Subtraktion.

Die Character-Nummer ergibt sich aus:

384-256=128.

Unser Pixel wird also innerhalb des Characters 128 dargestellt, und zwar durch jenen innerhalb des zweiten Blocks zu je 256 Charactern - denn einmal haben wir ja 256 abgezogen, weil die Zahl größer als 256 war.

Um die Adresse herauszufinden - denn die Character-Nummer ist eigentlich ohne Große Bedeutung, müssen wir ein bißchen anders rechnen.

Die untereinanderliegenden 8 Pixelzeilen werden jeweils von aufeinanderfolgenden Bytes in der Musterbeschreibungstabelle definiert. Eine ähnliche Speicherung haben Sie bereits in Verbindung mit dem Multicolor-Modus kennengelernt.

Das bedeutet, daß zunächst einmal festgestellt werden muß, zu welchem 8er-Block von Pixelspalten das Pixel gehört, das wir setzen möchten, wobei die Blocknumerierung von 0-23 geht, denn auf dem Bildschirm befinden sich ja 24 Blocks zu je 8 Pixelzeilen.

12/8=1 (Rest zunächst unbedeutend)

Pro 8er-Block müssen 256 Bytes addiert werden, denn diese nimmt jeder in Anspruch.

16=256.

Kümmern wir uns nun zunächst um die Spalte.

Für jede Spalte in einem 8-Byte-Block werden 8 Bytes benötigt. Deshalb müssen für jed Spalte zu den 256 Bytes, die sich aus der Zeilenkoordinate ergeben, weitere für die Spalte addiert werden.

256+35=291

Damit haben wir die Adresse, die unser Byte enthält. In diesem Byte ist jedoch nicht nur die Existenz von 97/35 gespeichert, sondern auch 97/27, 97/19, 97/11, 97/3, 97/43, 97/51, und 97/59.

Um nun an unser Bit zu kommen - denn genau ein Bit ist ja für die Darstellung eines Punktes auf dem Bildschirm verantwortlich - müssen wir uns näher mit den Koordinaten beschäftigen.

Zunächst teilen wir die Pixelspalte durch 8, um eine Characterspalte zu erhalten.

35/8=4 (Teilungsrest zunächst unbedeutend)

Dann teilen wir zunächst die Characterzeile, die wir errechnet haben (in ihr befindet sich unser Pixel), durch die Anzahl der Pixelzeile pro Characterzeile. Diesmal ist aber nicht das Ergebnis für uns wichtig, sondern der ganzzahlige Teilungsrest.

Bei der Teilung von 12/6 erhalten wir einen Rest von 4. Wenn wir künftig von einem Teilungsrest reden, sprechen wir von

12 MOD 8 = 4.

BROKER - DAS BÖRSENSPIEL

Nach soviel Lernen wird es Zeit, sich ein wenig zu unterhalten. Dazu ist da BÖRSENSPIEL von HAGERA(R) genau das Richtige!

Kaufen und verkaufen Sie Aktien, berufen Sie die Hauptversammlung ein, werden Sie Aufsichtsratsmitglied, hetzen Sie Ihren Gegenspielern den Fiskus auf den Hals!

Der Computer verwaltet alles blitzschnell, wertet Kursveränderungen grafisch aus und verteilt Dividenden.

Ein Spiel für Lange Abende: Jeder Spielstand kann abgespeichert und später wieder eingelesen werden. 2 oder 3 Spieler können gleichzeitig mitspielen - aber natürlich kann auch in Gruppen gespielt werden.

Das BÖRSENSPIEL sollte in Ihrer Sammlung nicht fehlen! Für TI-99/4a, XBasic, 32Kb.

BROKER mit ausführl. Handbuch nur DM 34,90 !!

TRAFFIC

Hugos Hund ist auf die Straße gelaufen, mitten zwischen den tosenden Verkehr. Sie als echter Tierfreund helfen natürlich sofort, den kleinen Ausreißer zu retten.

Die rüden Autofahrer indessen kümmert dies reichlich wenig: LKWs, PKWs, Dampfwalzen und andere Fahrzeuge trachten Ihnen nach dem Leben. Mit jeder Runde wird der Verkehr immer stärker!

TRAFFIC von HAGERA(R) kann auf dem TI-99/4a mit Extended Basic und Joysticks gespielt werden. Auch diesem Soiel ist eine ausführliche deutsche Anleitung beigefügt.
Auf Cassette oder Diskette (bitte angeben!)

TRAFFIC - Das rasante Actiongame DM 24.90 !!

Wenn Sie neben der hochauflösenden Grafik auch noch Farben wünschen, ist es erforderlich, auch das entsprechende Nybble der Farbtabelle zu ändern.

Die Berechnung, welches Nybble geändert werden muß, erfolgt im Wesentlichen wie die Berechnung für das Muster in der Musterbeschreibungstabelle.

Für das Setzen des Farbwertes für sichtbare Punkte ist es aber unerheblich, in welcher Pixelspalte genau sich der zu setzende Punkt befindet, da 8 nebeneinanderliegende Punkte jeweils eine Einheit bilden.

Von diesen 8 Punkten werden, wie bereits erwähnt, die sichtbaren durch das erste Nybble und die unsichtbaren Punkte durch das zweite Nybble des Bytes bestimmt, das parallel zu Byte in der Musterbeschreibungstabelle genausoweit von der Startadresse der Farbtabelle entfernt ist, wie das musterbeschreibende Byte zur Startadresse der Musterbeschreibungstabelle.

Soll hingegen der Farbwert der 'nicht sichtbaren' Punkte geändert werden, ist das zweite Nybble zu verändern. Beachten Sie dabei, daß durch Veränderung der Hintergrundfarbe eines Characters diese ebenfalls sichtbar werden, wenn die neue Farbe weder der Hintergrundfarbe noch Transparent entspricht.
so zu unangenehmen Farbeffekten kommen kann.

Noch ein Wort zu Sprites.

Wir haben bereits gesagt, daß es auch im BIT-MAP-MODUS möglich ist, Sprites zu benutzen. Nicht möglich ist dahingegen die Verwendung der automatischen Bewegung, da im VDP-Ram für die erforderliche Sprite-Bewegungstabelle aufgrund des notwendigen größeren Speicherplatzes von Farbtabelle und Muster-beschreibungstabelle kein Raum mehr vorhanden ist.

Wenn Sie Sprites verwenden und diese auch bewegen wollen, so müssen Sie sich um die Bewegung selbst kümmern. Entweder können Sie dies durch ein geeignetes Programm über den USER-INTERRUPT. Ein derartiges Programm ist jedoch sehr kompliziert und eine Erläuterung würde den Rahmen dieses Kurses sprengen. Einfacher und leichter Durchführbar ist eine Veränderung der Spriteposition durch die Bestimmung von neuen Positions- koordinaten in der Sprite-Attributen-Liste.

Sobald Sie in den jeweils beiden ersten Bytes eines 4-Byte-Blocks, welche für die X- und Y-Koordinaten ausschlaggebend sind, neue Werte festlegen, wird die Position des Sprites verändert.

An geeigneter Stelle im Programm untergebracht, erreichen Sie auch auf diese Art eine relativ schnelle und ruckfreie Bewegung; ähnlich, wie Sie sie vom Extended Basic her gewohnt sind.

Damit haben Sie nun alle vier Arbeitsmodi des TI-99/4a kennengelernt. Auf den folgenden Seiten finden Sie einige Übungsaufgaben, die zunächst nur den Bit-Map-Modus, dann aber auch die anderen Modi des TI-99/4a betreffen.

Wenn Sie mit einer Frage nicht sofort weiterkommen, lösen Sie zuerst die folgenden und schlagen Sie anschließend erst noch einmal im entsprechenden Kapitel nach, bevor Sie im Lösungsteil suchen.

Die folgenden beiden Kapitel setzen voraus, daß Sie über die Arbeitsmodi des TI-99/4a Bescheid wissen. Wir werden darin von diesen Modi reden, ohne nochmals genauer auf sie einzugehen. Lesen Sie daher erst weiter, wenn Sie die vorangegangenen Kapitel vollends beherrschen.

ÜBUNGSAUFGABEN

16. Auf welche Art und Weise können Sie die Bildschirmdarstellung des Bit-Map-Modus löschen?

- Durch Löschen der Bildschirmdarstellungstabelle
- Durch Löschen der Farbtabelle
- Durch Löschen der Musterbeschreibungstabelle
- Durch Löschen von VDP-Register 0
- Durch Setzen von VDP-Register 7 auf >00
- Durch Setzen von Adresse >83D4 auf >00

17. Welche der unter 1 genannten Arten ist am sinnvollsten für ein Löschen des Bildschirms, damit nach dem Löschen im Modus mit wirklich leerem Bildschirm weitergearbeitet werden kann?

18. Schreiben Sie eine Routine, welche den Bildschirm löscht. Die Farbtabelle beginnt bei >0000, die Bildschirmdarstellungs-Tabelle bei >1800 und die Musterbeschreibungstabelle bei >2000.

19. Wo müssen Sie eine Veränderung vornehmen, um im Bit-Map-Modus einen Punkt an eine bestimmte Stelle zu setzen.

20. Wo müssen Sie eine Veränderung vornehmen, um im Bit-Map-Modus einen Punkt mit einer bestimmten Farbe setzt.

21. Schreiben Sie eine kleine Routine, welche einen Punkt an einer bestimmten Stelle umfärbt, ohne Punkte zu löschen oder neue zu setzen.

22. Welche Koordinaten werden durch Byte 300 in der Musterbeschreibungstabelle beeinflusst?

23. Was bewirkt das folgende Programm:

```
       REF VSBW,VWTR
       DEF START
*
START  LI R0,>0003
       BLWP @VWTR
       LI R0,0207
       BLWP @VWTR
       LI R0,>1C00
LOOP1  CLR R1
LOOP2  SWPB R1
       BLWP @VSBW
       SWPB R1
       INC R1
       INC R0
       CI R1,>100
       JNE LOOP2
       CI R0,>300
       JNE LOOP1
*
LOOP3  NOP
       JMP LOOP3
```

24. Mit welchen Bytes können Sie folgende Koordinaten beeinflussen?

10/54, 103/22, 11/22, 99/80, 20/56, 204,190, 28/115.

25. Wir haben Ihnen ein Programm zum setzen von Punkten vorgestellt. Welche Bytes werden bei den folgenden Registerwerten in der Farbtabelle geändert?

R4=	10	103	11	99	20	204	28
R5=	22	80	54	115	190	80	22
R6=	5	14	0	11	8	3	1

26. Welche der folgenden Aussagen ist unrichtig?

a) Im Multicolor-Modus sind 24x40 Bildschirmpositionen verfügbar.

b) Im Grafikmodus sind 192x256 Bildpunkte verfügbar.

c) Im Textmodus sind keine Sprites verfügbar.

d) Im Multicolor-Modus sind Sprites verfügbar.

e) Die Farbtabelle im Grafikmodus beginnt an >0380 im VDP-Ram.

f) Innerhalb eines Programms ist es möglich, verschiedene Modi nacheinander zu verwenden.

g) Der Basic-Interpreter arbeitet im Text-Modus.

h) Der Multicolor-Modus ist besonders für Verwaltungsprogramme geeignet, namentlich für Textverarbeitung und Dateien.

i) Die Bildschirm-Darstellungstabelle ist im Bit-Map-Modus, im Multicolor-Modus und im Grafik-Modus gleichlang.

j) Die Musterbeschreibungstabelle ist im Multicolormodus, im Textmodus und im Bit-Map-Modus gleichlang.

k) Im Textmodus wird keine Farbtabelle benötigt.

l) Der folgende Programmteil löscht den Bildschirm des Text-Modus:

```
        LI R0,>0000
        LI R1,>2000
CLR     BLWP @VSBW
        INC R0
        CI R0,>300
        JNE CLR
```

m) Im Bit-Map-Modus sind Sprites, aber nicht deren automatische Bewegungsmöglichkeit verfügbar.

n) Im Textmodus wird ein Character in der Musterbeschreibungstabelle durch 6 Bytes definiert, da die Darstellung auf dem Bildschirm nur mit 6x8 Pixel und nicht wie im Grafik-Modus mit 8x8 Pixel erfolgt.

o) Im Grafik-Modus sind Sprites, aber nicht deren automatische Bewegungsmöglichkeit verfügbar.

p) Um den Bit-Map-Modus benutzen zu können, ist es erforderlich, VDP-Register 0 zu ändern und eine Kopie dieses Wertes an >83D4 zu schreiben.

q) Die Farbe der Character wird im Text-Modus durch Adresse >83D4 bestimmt.

r) Die Farbe des Bildschirms wird in allen Arbeitsmodi durch den Wert des niedrigwertigsten Nybble in VDP-Register 7 bestimmt.

27. Die Startadressen der verschiedenen Tabellen des VDP-Ram können durch Veränderung von bestimmten Registern geändert werden. Ordnen Sie die 3 Aussagegruppen zueinander.

a) - Bildschirmdarstellungstabelle
   - Farbtabelle
   - Sprite-Attributenliste
   - Musterbeschreibungstabelle
   - Sprite-Darstellungstabelle

b) - VDP-Register 2
   - VDP-Register 3
   - VDP-Register 4
   - VDP-Register 5
   - VDP-Register 6

...

c) Registerinhalt multipliziert mit 2048.
Registerinhalt multipliziert mit 2048.
Registerinhalt multipliziert mit 6.
Registerinhalt multipliziert mit 32.
Registerinhalt multipliziert mit 64.

28. Schreiben Sie ein Programm, welches folgende Bedingungen erfüllt:

a) Umschalten auf Tastendruck in einen beliebigen Modus.

b) Im Grafik- und Text-Modus die Möglichkeit, über die Tastatur den Bildschirm zu beschreiben und den Text zu behandeln, wie in den Übungsaufgaben zum Text-Modus beschrieben.

c) Im Multicolor-Modus die Möglichkeit, mit den Cursortasten einen Cursor zu einer beliebigen Bildschirmpositon zu bewegen und einen angesteuerten Punkt zu setzen oder zu löschen.

d) Im Bit-Map-Modus die gleiche Möglichkeit. Achten Sie auf die Farbzusammenhänge in den Pixelzeilen.

e) In jedem Modus muß eine geeignete Routine zum Löschen des Bildschirms zur Verfügung stehen.

f) Für Experten: Versuchen Sie einmal, eine Möglichkeit zu finden, auch im Bit-Map-Modus schnell und problemlos Texte darzustellen, ohne auf Sprites zurückgreifen zu müssen.

29. Welcher Modus ist am besten geeignet, die folgenden Probleme zu lösen?

a) Dateiprogramm b) Balkendiagramme
c) Schnittmusterzeichnungen
d) Grundrißzeichnungen
e) Spiele mit Farbe und Bewegung
f) Musikprogramm g) Grafik Tablett
h) Titelbilder mit möglichst großen Buchstaben

PROBLEMLöSUNGEN

Haben Sie Softwaresorgen???

Wir lösen Ihre Softwareprobleme

- zu günstigen Preisen

- auf unterschiedlichsten Computern

- auf allen Gebieten!

Melden Sie sich jetzt! Fordern Sie unsere Gratisinfo bezüglich AUFTRAGSARBEITEN an. Wir lösen Ihre Softwareprobleme!

Und auch dann, wenn Sie kein Programm schreiben lassen wollen, sind wir in Sachen COMPUTER für Sie die richtigen Ansprechpartner!

1.8

1.8. BASIC-ZUGRIFF AUF MASCHINENPROGRAMME

Einige von Ihnen haben sicher schon daran gedacht, wenn nicht sogar schon damit experimentiert: Der Aufruf von Maschinen-Programmen aus dem Basic. Sicher ist dem Einen oder Anderen schon der Gedanke gekommen, daß es doch möglich sein muß, aus dem Basic heraus ein Maschinenspracheprogramm anzusprechen und so bestimmte Abläufe, die in Basic viel Zeit brauchen, zu beschleunigen.

In diesem Kapitel wollen wir uns näher damit beschäftigen. Dazu müssen wir noch einmal auf den Unterschied zwischen Assembler und Basic eingehen.

Ein Assembler-Programm besteht aus einzelnen Befehlen, von denen jeder für sich im Prozessor des TI-99/4a eine ganz bestimmte Funktion veranlasst.

Ein Basic Programm besteht aus einzelnen Befehlen, von denen jeder für sich eine ganze Reihe von Funktionen im Prozessor veranlasst.

Jeder dieser Programmtypen nimmt einen bestimmten Platz im RAM, dem 'Random Access Memory', also dem Benutzerspeicher, ein. Damit ist klar, daß jedes Programm irgendwo eine Startadresse haben muß, also auch ein Basicprogramm.

Ein Basic-Programm wird mit RUN gestartet. RUN bewirkt die Bereitstellung von Speicherplatz für Variable, Arrays etc. und den Start des Programms an der Basiczeile, welche die niedrigste Zeilennummer hat. In Extended Basic ist es sogar möglich, diese Zeilenbezeichnung selbst anzugeben, zum Beispiel mit RUN 1000.

In Maschinensprache gibt es aber keine Zeilennummern. Stattdessen haben wir dort unsere 'Labels'. Das sind die Bezeichnungen, die in einer Befehlszeile des Assemblers vor der eigentlichen Mnemonic stehen.

```
10 PRINT "BASICPROGRAMM"
```

```
PRINT  LI R0,>0000
```

Hier sieht man deutlich den Unterschied. Die 'Eingangsstelle' des Basicprogramms ist die Zeilennummer (hier '10'), die Eingansstelle des Maschinenprogramms ist der Label PRINT.

Wenn wir also ein Maschinenprogramm starten wollen, müssen wir zu seiner Eingangsstelle verzweigen.

```
GOTO PRINT
```

gibt es aber nicht. Um ein Maschinenprogramm vom Basic her starten zu lassen, sind eine ganze Reihe weiterer Dinge zu beachten. Dazu müssen wir zunächst einmal unser LABEL näher betrachten.

In einem Maschinenprogramm kann es vordefinierte und nicht vordefinierte Eintrittsstellen geben. Als vordefiniert gelten alle, die am Programmstart in einer DEF-Instruktion genannt werden. Im folgenden Programm ist CLEAR vordefiniert und CL1 ist nicht vordefiniert.

```
       DEF CLEAR
       REF VSBW
*
CLEAR  LI R0,>0000
       LI R1,>2000
CL1    BLWP @VSBW
       INC R0
       CI R0,>0300
       JNE CL1
  .
  .
  .
```

Dieses Programm löscht, wie Sie warscheinlich richtig erkannt haben, den Bildschirm. Nachdem Sie es in Maschinencode umgewandelt haben, müssen Sie die Voraussetzungen für die Benutzung des Programms aus dem Basic schaffen.

Zum Vorbereiten, Laden und Ausführen von Maschinenprogrammen stellt das E/A-Modul-Basic und das Extended Basic Modul je drei Befehle zur Verfügung. Diese sind:

```
CALL INIT
CALL LOAD("Dateinamenbeschreibung")
CALL LINK("Programmname")
```

Mit INIT bereiten Sie den Rechner auf die Benutzung von Maschinenprogrammen vor. Es wird Speicherplatz in der Speichererweiterung reserviert; gleichzeitig werden alle Maschinenprogramme in der Erweiterung gelöscht und in Extended Basic auch einige 'Utilities' vorbereitet, die Sie noch kennenlernen werden. Sie sollten INIT einmal am Anfang ihres Basicprogramms setzen und später nicht mehr.

Mit LOAD laden Sie ein Maschinenprogramm von Diskette. Nehmen wir an, Sie hätten ein Programm unter dem Namen FILE gespeichert, und die Diskette befindet sich in Laufwerk 1, dann lautet der Ladebefehl:

```
CALL LOAD("DSK1.FILE")
```

LINK schließlich führt ein Maschinenprogramm aus, welches zuvor geladen werden muß. Wenn Sie das gezeigte Programm mittels LOAD in die Speichererweiterung geladen haben, führt der folgende Befehl das Programm aus:

```
CALL LINK("CLEAR")
```

Genauso, werden Sie sich sagen, müsste es funktionieren - aber denkste!!!

Ganz so einfach, wie es aussieht, ist es doch nicht. Ihr Rechner wird, wenn Sie alle vorgenannten Hinweise ausführen, so ziemlich alles machen - aber nicht den Bildschirm löschen und dann ein Basicprogramm weiter ausführen. Wie denn auch??? Unser TI-99/4a ist (leider) nicht allwissend, sondern trotz Maschinensprache auf das angewiesen, was Sie Ihm angeben.

Sicher haben Sie bisher alles richtig gemacht: Initialisiert, geladen und dann das Programm an einer vordefinierten Eingangsstelle ausgeführt. Trotzdem funktioniert es nicht - aber das hat - glauben Sie es ruhig - alles seinen Grund.

Um Ihnen zu zeigen, was wir in unserem kleinen Programm alles unterlassen haben, zäumen wir das Pferd erst einmal am Schwanze auf:

Der Rücksprung! Unser kleines CLEAR-Programm löscht den Bildschirm - und dann? Drei Punkte haben wir ans Ende des Programms gesetzt. Dies bedeutet, daß nach unserem JNE CL1 noch irgendetwas folgt.

Normalerweise beenden Sie in Maschinensprache einen Programmabschnitt, indem Sie irgendwohin verzweigen oder zu etwas zurückspringen. Beide Möglichkeiten können Sie verwenden, um aus einem Maschinenprogramm ins laufende Basicprogramm zurückzukehren.

RT (Return) springt zum aufrufenden Programm zurück, indem es eine Sprungadresse aus Register 11 holt. RT ist also nichts anderes als B . Demnach wäre also auch ein Rücksprung mit dieser Verzweigeinstruktion möglich. Genauso zum Ziel führt auch die Anweisung B @>0070, denn auch in >0070 befindet sich die Adresse des laufenden Programms, zu der zurückgesprungen werden muß.

Was aber, wenn wir nun Register 11 innerhalb unseres Maschinenprogramms verändern, weil wir es für eigene Zwecke benötigen?

Dann wäre in R11 nicht mehr die Rücksprungadresse, sondern irgendetwas gespeichert, und der Rücksprung erfolgt überall hin - nur nicht zum aufrufenden Programm.

Und wie ist es mit den anderen Registern? Wenn wir vom Basic aus ein Maschinenprogramm aufrufen, haben doch alle Register bestimmte Werte, die nach der Rückkehr ins Basic wieder benötigt werden. In den vorangegangenen Kapiteln haben Sie aber gesehen, wie wichtig Register für uns sind. Auf R0, R1 und R2 kann man so gut wie gar nicht verzichten.

Das bedeutet, wir müssen die Inhalte unserer 16 Register sichern, bevor wir in unserem Maschinenprogramm damit arbeiten.

Dazu könnten wir theoretisch alle Registerinhalte in irgendwelche Speicherstellen schieben, und zwar durch MOV. Dies kostet aber Zeit und Speicherplatz. Einfacher wäre es, wenn man sich seine eigenen Register definieren könnte, und den Wert der Register, die im Basic-Programm benötigt werden, überhaupt nicht antastet.

LWPI EIGENE veranlasst den Computer dazu, an der Speicherstelle EIGENE einen Registerbereich beginnen zu lassen, den wir dann im Maschinenprogramm wie gewohnt mit R0 bis R15 ansprechen können. Natürlich müssen wir vorher an der Stelle EIGENE dafür Speicherplatz reservieren. Dies geschieht mit unserem Befehl zur Bytereservierung, BSS.

EIGENE BSS >20 reserviert 32 Bytes Speicherplatz für unsere eigenen Register. Da jedes Register ein 2-Byte-Wort ist, reicht dieser Platz aus, denn 16*2=32 (>20)

Zweckmäßigerweise starten wir EIGENE an einer Stelle im Speicher, die ohnehin für unsere eigenen Register vorgesehen ist. Diese Stelle wird gekennzeichnet durch die vordefinierte Stelle USRWS und befindet sich bei >202A.

Wir schreiben also:

```
 .
 .
 .
USRWS  EQU  >20BA
EIGENE EQU  USRWS
       BSS  >40
 .
 .
 .
```

Natürlich müssen wir, bevor wir aus dem Maschinenprogramm ins Basic zurückkehren, wieder die dort benutzten Register laden. Um dies zu tun, setzen wir unmittelbar vor den Rücksprung den Befehl:

```
LWPI >83E0
```

In dieser Adresse ist die Startadresse der Register aller sogenannten Konsolenprogramme - also zum Beispiel des Basic-Interpreters.

Schauen wir uns jetzt die Veränderungen unseres Maschinenprogramms einmal an:

```
       DEF CLEAR
       REF VSBW
*
USRWS  EQU >20BA
EIGENE EQU USRWS
       BSS >40
*
CLEAR  LWPI EIGENE
       LI R0,>0000
       LI R1,>2000
CL1    BLWP @VSBW
       INC R0
       CI R0,>0300
       JNE CL1
       LWPI >83E0
       RT
       END
```

Das sieht doch schon ganz anders aus - und es lässt sich sogar etwas vereinfachen. USRWS lässt sich natürlich als eigener Registerbereich auch selbstätig ansprechen:

```
       DEF CLEAR
       REF VSBW
*
USRWS  EQU >20BA
GPLREG EQU >83E0
*
CLEAR  LWPI USRWS
       LI R0,>0000
       LI R1,>2000
CL1    BLWP @VSBW
       INC R0
       CI R0,>0300
       JNE CL1
       LWPI GPLREG
       RT
       END
```

Leider werden Sie feststellen, daß auch dieses Programm nicht zum gewünschten Erfolg führt. Zuvor müssen wir nämlich noch das GPL-Status-Byte (nicht zu verwechseln mit dem Statusregister, welches vom Programmablauf beeinflusst wird) löschen, um es für seine weitere Verwendung im Basic vorzubereiten. Das GPL-Status-Byte befindet sich an Adresse >837C, und diesen Löschvorgang schieben wir noch zwischen das eigentliche Maschinenprogramm und das Laden der alten Register.

Damit nimmt unser Programm langsam Formen an:

```
       DEF CLEAR
       REF VSBW
*
USRWS  EQU >20BA
GPLREG EQU >83E0
GPLSTA EQU >837C
*
CLEAR  LWPI USRWS
       LI R0,>0000
       LI R1,>2000
CL1    BLWP @VSBW
       INC R0
       CI R0,>0300
       JNE CL1
*
       CLR R0
       MOVB R0,@GPLSTA
       LWPI GPLREG
       RT
*
       END
```

Auch das derart erweiterte Programm führt noch nicht zum Ziel, obwohl scheinbar alles stimmt. Leider hat aber Texas Instruments vor den Erfolg das VDP-Ram gestellt. In diesem Ram werden bekanntlich die Tabellen und Listen für Bildschirmausgabe etc. untergebracht. In Maschinensprache wird die gesamte Tabelle nur zu diesem Zweck benutzt. Wenn Sie hingegen in Basic oder Extended Basic programmieren, befinden sich in diesem Speicherbereich auch Basicprogramme.

Zu Recht werden Sie sich jetzt sicher fragen, was denn das Basicprogramm dort zu suchen hat. Eigentlich nichts, stimmt! Doch dadurch, daß verschiedene Tabellen im Basic die gleiche Startadresse haben, werden über 1.5 Kbyte RAM für Basicprogramme gewonnen - für den reinen Basic-Programmierer sicher eine gute Sache.

Wer aber Hilfsroutinen aus dem Basic/Extended Basic heraus ansprechen will, stößt dadurch auf Hindernisse. So wird in den beiden Basic-Versionen die Startadresse für die Musterbeschreibungstabelle und die Bildschirmdarstellungstabelle an der gleichen Stelle festgelegt. Da die ersten 768 Bytes durch die Bildschirmdarstellung benutzt werden, müssen die Characterdefinitionen zwangsläufig am Ende dieser Tabelle beginnen. Daraus resultiert die Begrenzung der Verfügbaren Zeichen im Basic.

Bei allen Ausgaben auf den Bildschirm ist es daher erforderlich, zum zu schreibenden ASCII-Wert >60 (=96) zu addieren, denn 96x8 (jede Musterbeschreibung benötigt 8 Byte) ergibt 768, und dies bezeichnet die erste freie Bytestelle hinter der Bildschirmdarstellung.

Deshalb muß im Register 1 der zu schreibende Wert geändert werden. Statt

```
LI R1,>2000
```

muß es heißen:

```
LI R1,>8000
```

Sie werden sich jetzt sicher fragen, was Sie mit einem solchen Programm anfangen können. Schließlich stellt Ihr Basic doch den Befehl CALL CLEAR zur Verfügung, mit dem es ohne weiteres möglich ist, den Bildschirm zu löschen - und man braucht sich nicht mit irgendwelchen Rücksprüngen herumzuschlagen.

Das ist sicher richtig. Jedoch eignet sich gerade dieses kleine, überschaubare Löschprogramm dazu, die Probleme eines MC-Aufrufs aus dem Basic heraus zu erkennen. Es gibt aber durchaus Fälle, in denen sich das Schreiben eines Maschinenprogramms lohnt.

Nehmen wir dazu einmal an, Sie möchten nur ein "Fenster" des Bildschirms löschen, als Beispiel von Koordinate 5/5 bis zur Koordinate 10/10. Dies kann der Fall sein, wenn die Umgebung immer sichtbar lassen wollen, aber im Fenster verschiedene Darstellungen wünschen. Dann wäre es doch sinnvoll, ein Programm

zu haben, welches nur dieses Fenster löscht.

In Basic ist dazu eine Schleife erforderlich:

```
10 FOR I=5 TO 10
20 CALL HCHAR(I,1,32,5)
30 NEXT I
```

Sicher ist diese Basicschleife noch relativ schnell. Trotzdem ist bereits ein Zeitunterschied zu verzeichnen, wenn Sie stattdessen folgendes eingeben, wobei die Zeit für das Laden des Maschinenprogramms (INIT/LOAD) nicht berücksichtigt wird, da Sie sicher später einmal mehrere MC-Programme gleichzeitig laden, wodurch diese Ladezeit wieder ausgeglichen wird.

Basic:

```
30 CALL LINK("CLFEN")
```

Assembler:

```
       DEF CLFEN
       REF VSBW
*
USRWS  EQU >20BA
GPLREG EQU >83E0
GPLSTA EQU >837C
*
CLFEN  LWPI USRWS
       LI R5,>0084
       LI R1,>2000
       MOV R5,R0
CL2    AI R5,5
CL1    BLWP @VSBW
       INC R0
       C R0,R5
       JLE CL1
       AI R5,>16
       CI R5,>012A
       JLE CL2
*
  .
  .
  .
```

```
.
.
.
       CLR R0
       MOVB R0,@GPLSTA
       LWPI GPLREG
       RT
*
       END
```

Sicher haben Sie schon Probleme gehabt, bei dem Sie sich wünschten, auf ein schnelles Maschinenprogramm zurückgreifen zu können - zum Beispiel das Definieren von Sonderzeichen in einem Spiel. Solche Definitionen brauchen in Basic sehr lange, sind aber in Assembler binnen Sekundenbruchteilen passiert.

Wenn Sie einige Programme in Assembler geschrieben haben, die aus dem Basic aufrufbar sind, werden Sie sicher irgendwann den Wunsch verspüren, diese zu verbessern. Zum Beispiel unser Fenster-Löschprogramm: Ist es nicht möglich, dieses so zu gestalten, daß die Werte 'variabel' sind und vom Basic ans Maschinenprogramm übergeben werden können?

Bevor wir Ihnen im nächsten Kapitel zeigen, wie man Parameter im Basic benutzt, sollten Sie jedoch die folgenden Übungsaufgaben lösen. Im Kapitel PARAMETERÜBERGABE gibt es nämlich wieder viel Neues - und ohne die hier genannten Grundlagen ist es nur schwer verständlich.

ÜBUNGSAUFGABEN

1. Mit welchem Befehl wird ein Maschinenprogramm im Basic aufgerufen:

- INIT
- LOAD oder
- LINK ?

2. Was müssen Sie alles tun, um ein Maschinenprogramm aus dem Basic heraus starten zu können?

3. Schreiben Sie ein Programm, welches den deutschen Zeichensatz zur Verfügung stellt, und machen Sie das Programm vom Basic her zugänglich!

4. Wie lautet die Startadresse für die Register, die von den Konsolenprogrammen (z.B. Basic-Interpreter), benutzt werden?

- >83D4
- >83E0
- >837C oder
- >20BA ?

5. Was verbirgt sich hinter dem Symbol 'USRWS'

6. Nennen Sie drei Arten, aus einem Maschinenprogramm ins Basic zurückzuspringen.

7. Schreiben Sie ein Programm, welches in den TEXT-Modus umschaltet und den Satz 'ICH LERNE ASSEMBLER' in die Mitte des Bildschirms schreibt. Das MC-Programm soll vom Basic aufgerufen werden. Das Basicprogramm selbst soll nach einem Tastendruck in ein weiteres Maschinenprogramm verzweigen und den Bildschirm löschen, um sodann in den Grafik-Modus zurückzukehren.

8. Womit wird verhindert, daß nach einem Maschinenprogrammaufruf das Register 11 verändert wird, welches die Rücksprungadresse enthält?

9. Was bewirkt 'EQU'?

10. Was bewirkt 'BSS'?

11. Was bewirkt 'LWPI'?

12. Welche der folgenden Aussagen ist wahr?

a) 'USRWS EQU >20BA' lädt die Startadresse des Benutzer-Arbeitsregisters an Adresse >20BA.

b) 'HIER BSS >20 reserviert einen 32 Byte großen Raum im Ram, beginnend bei Adresse HIER.

c) 'CALL LOAD("DSK2.START")' lädt ein Maschinenprogramm von Diskettenlaufwerk 2, welches eine vordefinierte Eingangsstelle mit dem Namen START enthält.

d) 'CALL LINK("START")' ruft ein Maschinenprogramm mit dem Namen START auf, welches zuvor mit LOAD geladen wurde.

e) 'CALL INIT' löscht Maschinenprogramme in der Speichererweiterung.

f) 'B >0070' ist ein gültiger Rücksprung.

g) 'B *R11' ist ebenfalls ein gültiger Rücksprung.

h) 'B DORT' ist ebenfalls ein gültiger Rücksprung, wenn in Adresse DORT und DORT+1 die Rücksprungadresse enthalten ist.

13. Was verstehen Sie unter ASCII-Offsett?

14. Wie können Zeichen auf dem Bildschirm dargestellt werden, wenn Sie ein Maschinenprogramm vom basic her aufrufen?

BASIC GRAFIK EXPANSION

Wollten Sie nicht immer schon mehr mit Computer und Drucker machen als Texte erstellen?

Bisher scheiterte die Sache daran, daß man auf eine Hardcopy in Basic getrost eine Viertelstunde warten mußte.

Doch jetzt gibt es die

BASIC GRAFIK EXPANSION

und diese produziert verschiedene Hardcopys in Sekundenschnelle. Auch die Generierung von neuen Druckerzeichen wird unterstützt. Alles zusammen mit 4 neuen Maschinenbefehlen für TI-99/4a in Verbindung mit einem epson-kompatiblen Drucker. Für:

Konfiguration mit Extended Basic DM 39.90
Konfiguration mit Editor/Assemb. DM 34.90

1.9

1.9. PARAMETERÜBERGABE BASIC <--> MC

Wenn Sie Maschinenprogramme aus dem Basic heraus ansprechen wollen, haben Sie das bisher durch einfachen Aufruf mit dem Namen getan. Etwas komplizierter wird die Sache, wenn Sie Werte, welche Sie im Basic verwenden, zur weiteren Bearbeitung an ein Maschinenprogramm weitergeben wollen.

Numerische und Stringvariable sind an einer ganz bestimmten Stelle im Speicher abgelegt. Wenn diese Werte nun im Maschinenprogramm verwendet werden sollen, müssen Sie dem Computer mitteilen, wo sich eine Variable befindet, von welchem Typ sie ist und ob Sie vom Basic ins Maschinenprogramm oder umgekehrt übertragen werden soll.

Um dies zu bewerkstelligen, hat der TI-99/4a eine Reihe von Routinen zur Verfügung. Sie können diese Routinen nutzen, indem Sie die Maschinenprogrammutilities, die unter dem Namen BSCSUP auf Diskette Part A des Editor/Assembler Pakets gespeichert sind, vor der Benutzung Ihres Maschinenprogramms laden. Ferner ist es erforderlich, die Programmbezeichnungen der benötigten Utilities im Maschinenprogramm mittels REF einzuschließen, bevor Sie das Programm assemblieren.

Die Programme funktionieren ähnlich, wie Sie es bereits von VSBW, VSBR, VMBW, VMBR und VWTR her gewohnt sind. Die nötigen Daten werden in den Arbeitsregistern übergeben; auch diesmal werden wieder die Register 0 bis 2 benutzt, um die restlichen für eigene Anwendungen zur Verfügung zu haben.

Denken Sie aber daran, daß diese Utility-Programme nicht von vornherein implementiert sind (jedenfalls nicht im Editor/Assembler Modul), sondern zur Benutzung erst von Diskette geladen werden müssen, bevor Sie Ihr Maschinenprogramm laufen lassen.

BSCSUP umfasst folgende Programme:

NUMREF
übergabe numerischer Parameter vom Basic an das MC-Programm.

NUMASG
übergabe numerischer Parameter vom MC-Programm an das Basic.

STRREF
übergabe von Stringparametern vom Basic an das MC-Programm.

STRASG
übergabe von Stringparametern vom MC-Programm an das Basic.

ERR
Verkettung mit dem Fehlermeldungsprogramm des Basic-Interpreters.

Das letzte Programm, ERR, wollen wir erst einmal zurückstellen. Es ist nur wichtig, wenn wir die übergebenen Parameter auf Fehler überprüfen wollen, aber das muß nicht sein. Eine entsprechende Sicherung kann ja auch im Basic-Programm untergebracht werden.

Beschäftigen wir uns zunächst näher mit der übergabe von Stringparametern. Diese können wesentlich einfacher gehandhabt werden als numerische Parameter, für die noch weitere Informationen erforderlich sind.

STRREF übergibt, wie bereits erwähnt, Stringparameter an ein Maschinenprogramm. Um dies zu tun, muß in Ihrem Basicprogramm einer der folgenden Befehlstypen enthalten sein:

```
CALL LINK("NAME",Stringkonstante)
CALL LINK("NAME",Stringvariable)
CALL LINK("NAME",Stringausdruck)
CALL LINK("NAME",Stringordnung)
```

Gehen wir einmal davon aus, daß Sie ein Maschinenprogramm schreiben möchten, welches in die Mitte des Bildschirms ein Wort schreibt, welches nicht durch das MC-Programm, sondern im Basic festgelegt werden soll. In diesem Fall wäre LINK vom 1. Typ - die Übergabe einer Stringkonstanten.

Um STRREF in diesem Fall zu benutzen, müssen Sie die Register 0 bis 2 wie folgt laden:

R0 - Inhalt=0 (ist immer 0, außer bei übergabe einer Ordnung)

R1 - Inhalt=1 (Parameterposition im LINK-Befehl. Der 'nullte' Parameter ist der Programmname, der erste die Stringkonstante).

R2 - Inhalt=ADR (Adresse, an welche Sie den String einlesen möchten).

Angenommen, folgender Befehl ist in Ihrem Basic-Programm enthalten:

```
CALL LINK("START","DAS IST EIN STRING-PARAMETER, KONSTANT")
```

In diesem Fall müssten die folgenden Teile in Ihrem Maschinenprogramm enthalten sein, um den Satz etwa in die Mitte des Bildschirms zu schreiben:

```
 .
 .
 .
       DEF START
       REF VMBW,STRREF
*
ADR    BSS >FF
*
START  CLR R0
       LI R1,>0001
       LI R2,ADR
       BLWP @STRREF
*
       LI R0,>0180
       LI R3,ADR
       INC R3
       MOV @ADR,R2
       ANDI R2,>FF00
NEXT   MOVB *R3+,R1
       AI R1,>6000
       BLWP @VSBW
       INC R0
       AI R2,>0100
       CB R2,@ADR
       JLE NEXT
 .
 .
 .
```

Nicht berücksichtigt sind dabei die Vorbereitungen für den Sprung ins Maschinenprogramm und für den Rücksprung; beides wurde im vorangegangenen Kapitel ausführlich erläutert.

Am Programmanfang sehen Sie die Eingliederung des STRREF Programms durch REF. Im Anschluß daran wird ein Buffer freigehalten, welcher eine Länge von 256 Bytes hat. Die maximale Stringlänge ist 255, denn im ersten Byte des Buffers wird die Länge des Strings angegeben, und ein Byte kann Zahlen zwischen 0 und 256 darstellen (256-1 für die Darstellung = 255).

Register 0 wird auf >0, Register 1 auf >1 gesetzt, denn es handelt sich nicht um eine Ordnung, und der zu übergebende Parameter befindet sich an Position 1. In Register 2 laden wir die Adresse des Buffers, die mit ADR bezeichnet ist.

Nach Durchführung von STRREF befindet sich der String aus dem LINK-Unterprogramm an Adresse ADR, wobei das erste Byte das Längenbyte und die folgenden die ASCII-Werte der Stringzeichen darstellen.

Diesen String jetzt an den Bildschirm weiterzugeben, stellt nur ein kleines Problem dar. VMBW können wir wegen des benötigten OFFSETS nicht verwenden. Daher benötigen wir eine art Schleife, welche Zeichen für Zeichen um >60 erhöht (erstes Byte im Register 1) und dieses durch VSBW ausgibt. Wir laden dazu Register 0 mit der Bildschirmposition (1.Spalte, Zeile 13) und Register 1 mit der Adresse, die unseren String enthält. Dazu müssen wir 1 addieren, denn das Längenbyte soll ja nicht mit ausgegeben werden.

Wenn Sie anstelle einer Stringkonstanten im Basicprogramm eine Variable oder einen Ausdruck angeben, verhält es sich genauso.

Ein wenig anders ist es bei Ordnungen. Sie können nicht einen Wert wie zum Beispiel A$(7,15) übergeben, wohl aber die gesamte Ordnung A$. Dies erreichen Sie, indem Sie Register 0 auf die Elementenzahl setzen. Im LINK-Unterprogramm des Basics übergeben Sie eine Ordnung durch Angabe des Variablennamens und der Dimension. Die Dimension wird angegeben durch eine entsprechende Anzahl Kommata in Klammern hinter dem Namen. Eine zweidimensionale Ordnung wie A$ wird wie folgt übergeben:

```
CALL LINK("START",A$(,))
```

Die Elementenzahl, die im Maschinenprogramm im Register 1 angegeben werden muß, kann auf zwei unterschiedliche Arten bestimmt werden. Ausschlaggebend ist, ob die Dimensionsbasis im Basic auf 0 oder 1 - durch Benutzung des Befehls OPTION BASE - gesetzt worden ist. Normalerweise ist die Dimensionsbasis 0, das heißt, alle Dimensionen beginnen mit dem Element Nummer 0.

Wenn die Dimensionsbasis 0 ist, errechnet sich die Elementenzahl aus der maximalen Zahl der Elemente minus 1. In unserem Beispiel ist A$ als 7x15 Array dimensioniert. Wenn wir jetzt auf Element A$(3,8) zugreifen möchten, dann müssen wir wie folgt vorgehen:

```
Dimension Nr.:
      1       2       3       4       5       6       7
-------------------------------------------------------
Wert  8       3       -       -       -       -       -
= 11  8   +   3
-------------------------------------------------------
   8 0+3 1    (2       3       4       5       6)
=      8   +  33
= 41
-------------------------------------------------------------------
```

Der Wert von Register 0 muß 41 sein.

Wenn Sie Strings vom Maschinenprogramm ans Basic zurückgeben möchten, müssen Sie genau umgekehrt verfahren. Zuerst ist der String im RAM zu entwickeln, und dieser dann mittels STRASG zu übergeben. Hier die wesentlichen Schritte:

```
 .
 .
       DEF START
       REF STRASG
*
SATZ   BYTE >13
       TEXT 'DIES IST EIN STRING'
*
START  ...
       CLR R0
       LI R1,SATZ
       LI R2,1
       BLWP @STRASG
 .
 .
```

Auch in diesem Beispiel haben wir die Vorbereitungen für den Aufruf und den Rücksprung nicht berücksichtigt, da diese bereits ausführlich erläutert wurden.

SATZ ist die Adresse, die unseren String enthält. Das erste Byte beinhaltet die Stringlänge, die folgenden Bytes den String im ASCII-Code. Daraus ergibt sich eine Speicherbelegung wie folgt, wenn wir annehmen, das SATZ = Adresse >A100 ist.

```
>A100 >13      (Offset hier nicht berücksichtigt!)
>A101 >44 D
>A102 >49 I
>A103 >45 E
>A104 >53 S
>A105 >20
>A106 >49 I
>A107 >53 S
>A108 >54 T
>A109 >20
>A10A >45 E
>A10B >49 I
>A10C >4E N
>A10D >20
>A10E >53 S
>A10F >54 T
>A110 >52 R
>A111 >49 I
>A112 >4E N
>A113 >47 G
```

Im Teil START, in den noch die Vorbereitungen für die Übergabe de Programmkontrolle (wie im vorangegangenen Kapitel beschrieben) zu treffen sind, erfolgt die Übertragung des Strings durch Setzen von R0 auf 0 (es handelt sich nicht um eine Ordnung), von R1 auf SATZ (Adresse im RAM, die den String beinhaltet) und R2 auf 1 (dies beschreibt die Parameterposition im LINK-Befehl des Basics).

Mit dem Befehl

```
CALL LINK("START",Stringvariable)
```

erhalten Sie, soweit das Maschinenprogramm (und die BSCSUP-Utilities von Diskette Part A des E/A-Pakets) geladen wurde, den String 'DIES IST EIN STRING' in der Variablen.

```
CALL LINK("START",C$)
PRINT C$
```

führt daher zur Ausgabe des im Maschinenprogramm entwickelten Strings auf dem Bildschirm.

Sie können nun zum Beispiel ein Maschinenprogramm schreiben, mit dessen Hilfe es möglich ist, über die Tastatur Zeichen einzugeben, die in einem Buffer registriert werden, und den Bufferinhalt anschließend an das Basic zurückgeben.

Wenn Sie eine Ordnung übergeben wollen, müssen Sie ebenfalls wie unter STRREF beschrieben die Elementenzahl errechnen und diese im Register 1 an STRASG übergeben.

Beachten Sie, daß im Basic durch STRASG angesprochene Verkettungen mit LINK an der bezeichneten Parameterposition aus diesen Gründen nur eine Stringvariable oder eine Ordnung enthalten darf. Dabei besteht die Ordnungsbezeichnung aus dem Namen, den Klammern und einer Anzahl Kommata, die der Ausdehnung entspricht; für ein zweidimensionales Stringarray also beispielsweise:

```
CALL LINK("START",A$(,))
```

Bevor wir uns nun mit der Übergabe numerischer Parameter beschäftigen, hier ein paar Übungsaufgaben, die Sie nach der Lektüre dieses ersten Abschnitts beherrschen sollten.

ÜBUNGSAUFGABEN

1. Welche der folgenden Aussagen sind wahr?

a) Mit STRREF werden Parameter aus dem Basic an das Maschinenprogramm übergeben.

b) Mit STRASG werden Parameter aus dem Basic an das Maschinenprogramm übergeben.

c) Es ist möglich, mit STRREF einen Stringausdruck zu übergeben.

d) Es ist möglich, mit STRASG einen Stringausdruck zu übergeben.

e) Register 0 muß bei STRREF und STRASG die Elementenzahl enthalten, wenn es sich um eine Ordnung handelt. Andernfalls ist der Wert von R0 unbedeutend.

f) Der Wert von R0 muß immer 0 sein, bevor STRREF oder STRASG aufgerufen werden.

g) Register 1 gibt die RAM-Adresse an, an welcher der String, der zu übergeben ist, sich befindet.

h) Wenn Register 1 ADR enthält, startet der String mit seinem ersten ASCII-Character an ADR+2.

i) Register 2 bezeichnet die Parameterpositon, an welche der String übergeben oder von welcher er übernommen werden soll.

j) Der größtmögliche Wert in Register 2 vor STRREF oder STRASG ist 16.

k) Nur wenn die BSCSUP-Utilities geladen sind, können Sie mit dem Editor/Assembler-Modul die Routinen STRASG und STRREF benutzen.

l) Die maximale Stringlänge, die übergeben werden kann, ist 256.

2. Welche der folgenden Utility-Routinen sind in BSCSUP nicht enthalten:

- STRREF
- NUMASG
- ERR
- VWTR
- VMBW
- STRASG
- XMLLNK
- NUMREF
- VSBW

3. Was bewirkt der folgende Programmteil, wenn Sie im Basic CALL LINK("START","WORT") durchlaufen?

```
 .
 .
 .
       DEF START,WORT
       REF VMBW,VSBW,STRASG,STRREF
*
SATZ   BYTE >04
       BSS >04
*
WORT   ...
       LI R0,>0000
       LI R1,>8000
W1     BLWP @VSBW
       INC R0
       CI R0,>0300
       JNE W1
*
 .
 .
```

Nächste Seite Fortsetzung.

```
 .
 .
 .
START  ...
       CLR R0
       LI R1,SATZ
       INC R1
       LI R2,SATZ
       BLWP @STRREF
       LI R0,>0000
       LI R2,4
       LI R3,SATZ
       INC R3
NEXT   MOVB *R3+,R1
       AI R1,>6000
       BLWP @VSBW
       INC R0
       DEC R2
       JNE NEXT
 .
 .
 .
```

4. Was müssen Sie tun, um eine Ordnung vom Basic an ein Maschinenprogramm zu übergeben?

5. Berechnen Sie die Elementenzahlen für die folgenden Elemente aus der Ordnung F$(10,5,3)! Die im basic bestimmte Dimensionsbasis (OPTION BASE) ist 0.

- F$(8,1,1)
- F$(5,5,2)
- F$(9,0,3)
- F$(2,2,2)
- F$(10,5,3)

6. Berechnen Sie die Elementenzahl zu den in Aufgabe 5 genannten Elementen aus der gleichen Ordnung für OPTION BASE 1.

7. Wie übergeben Sie einen Ausdruck im Basic vom Basic an ein Maschinenprogramm.

8. Was bewirkt die folgende Basic-Anweisung:

```
CALL LINK("PROG",C$,"DOMINO")
```

9. Schreiben Sie ein Maschinenprogramm, welches einen String vom Basic übernimmt und verkehrt herum wieder an das Basic zurückgibt. Zum Beispiel soll der Basic-Befehl

```
CALL LINK("START","REGEN",C$)
```

soll anschließend in C$ "NEGER" enthalten.

10. Schreiben Sie ein Programm, welches auf dem Bildschirm den übergebenen String senkrecht in die Mitte schreibt.

Für die Übergabe von numerischen Parametern werden wir noch ein wenig weiter ausholen müssen. Dennoch wollen wir alle Schwierigkeiten, die auftreten, erst einmal zurückstellen und uns nur mit dem Utility-Routinen NUMREF und NUMASG beschäftigen.

Beide Routinen arbeiten im Prinzip genauso, wie Sie es von den String-Routinen her gewohnt sind. Beschäftigen wir uns zunächst näher mit NUMREF.

Sie können einen numerischen Parameter auf eine der folgenden Arten übergeben:

```
CALL LINK("NAME",numerische Konstante)
CALL LINK("NAME",numerische Variable)
CALL LINK("NAME",numerischer Ausdruck)
CALL LINK("NAME",numerische Ordnung)
```

Gehen wir einmal davon aus, Sie möchten ein Maschinenprogramm schreiben, welches das Ergebnis eines numerischen Ausdrucks an eine bezeichnete Position des Bildschirms schreibt. Der Aufruf aus dem Basic soll mit der Zeile

```
CALL LINK("ARITHM",142,2*5-3)
```

erfolgen, wobei es freigestellt ist, anstelle von 142 einen anderen Bildschirmpunkt von 0-760 und anstelle des Ausdrucks einen anderen mit einem Wert zwischen -32768 und 32767 zu wählen.

Für die Übernahme der Parameter müssen Sie NUMREF verwenden, und zwar auf folgende Art und Weise:

R0 - Inhalt=0 (ist immer 0, außer bei Übergabe einer Ordnung)

R1 - Inhalt=1 (für die Übernahme der Bildschirmposition, denn die Parameterposition dieses Wertes ist 1)

=2 (für die Übernahme des numerischen Ausdrucks, denn die parameterposition des Ausdrucks ist 2)

R2 wird von NUMREF und NUMASG nicht benutzt.

Die Adresse, an welche die Parameter übergeben werden, ist >834A.

Diese eine Adresse würde jedoch für die Aufnahme der Zahl nicht ausreichen. Deshalb sind auch die folgenden bis >835B für die Übergabe und Übernahme des numerischen Parameters benutzt.

Schauen wir uns dazu zunächst einmal an, wie unsere Bildschirmposition ins Maschinenprogramm gelangt.

```
  .
  .
  .
       DEF ARITHM
       REF VSBW,NUMREF
*
ARITHM ...
       LI R0,0
       LI R1,1
       BLWP @NUMREF
  .
  .
  .
```

Die '142' befindet sich jetzt im RAM, beginnend an Adresse >834A, und zwar folgendermaßen:

>834A >41
>834B >0E
>834C >02
>834D >00
>834E >00
>834F >00
>8350 >00
>835A >00

Sie werden sich jetzt erstaunt fragen, wo sich denn die 142 befindet, da sie ja offensichtlich auch trotz größter Bemühungen nicht aus dem Speicherbereich, der bei >834A beginnt, herausgelesen werden kann.

Das Geheimnis liegt in der sogenannten RADIX 100 SCHREIBWEISE. In dieser Art werden alle Zahlen in Exponentialform dargestellt. Der Exponent ergibt sich aus der Adresse >834A, wenn man ihn um >40 vermindert.

Für die Darstellung der 142 schreiben wir die Werte der folgenden Bytes, umgerechnet in Dezimalzahlen, einfach einmal nebeneinander, wobei die Adressen, die selbst und deren alle folgenden Null enthalten, nicht berücksichtigt werden.

>0E = 14
>02 = 2

14...2

Daraus ergibt sich 142.

Diese multiplizieren wir nun mit '10 hoch 1' (der 1, die wir in unserer Exponenten-Adresse übrigbehalten haben):

142 x 10 hoch 1 = 142 x 1 = 142.

Eine so dargestellte Zahl lässt sich vom TI jedoch noch nicht weiter verarbeiten. Wir müssen dazu ein weiteres Utility Programm bemühen, welches wir bisher noch nicht angesprochen haben.

Es handelt sich dabei um XMLLNK.

Diese Routine ist, ähnlich wie VSBW, VMBW, VSBR, VMBR und VWTR, bereits in der Konsole, also im Editor/Assembler-Modul, enthalten. In dieser Utility-Routine stehen verschiedene Teilroutinen mit unterschiedlichen Aufgaben zur Verfügung. Zwei davon sind wesentlich für die Benutzung unseres Stapelspeichers für die Parameter, der an Adresse >834A beginnt.

Um dieses Utilities nutzen zu können, müssen Sie REF XMLLNK an den Anfang Ihres Programms stellen.

Die beiden Routinen, die wir benutzen werden, sind:

```
CFI     EQU >1200
CIF     EQU >2300
```

CFI verwandelt eine Fließkommazahl, wie wir Sie in >834A ff. haben, in eine Integer-Zahl. die sich in >834A und >834B befindet (also Werte zwischen 0 und >FFFF haben kann, was die genannte Beschränkung der Werte auf -32768 und 32767 begründet).

CIF bewirkt im umgekehrten Fall, daß ein Wert der sich im Stapelspeicher an >834A und >834B befindet, in eine Fließkommazahl umgewandelt wird.

Die Startadressen der jeweils gewünschten Routine sind unmittelbar nach dem Aufruf von XMLLNK in einer DATA-Zeile anzugeben.

Um nicht immer mit >834A arbeiten zu müssen, geben wir dieser wichtigen Adresse einen Namen:

```
FAC     EQU >834A
```

Unser Programm erweitert sich damit wie folgt:

```
 .
 .
 .
       DEF ARITHM
       REF VSBW,NUMREF,XMLLNK
*
FAC    EQU >834A
CFI    EQU >1200
*
ARITHM ...
       LI R0,0
       LI R1,1
       BLWP @NUMREF
       BLWP @XMLLNK
       DATA CFI
 .
 .
 .
```

Den Wert, der sich jetzt in FAC und FAC+1 befindet, müssen wir sichern, denn wir wollen ja noch weitere numerische Parameter einlesen, die dann dieselben Adressen benutzen.

Wir benötigen also eine Adresse, an die wir den Wert aus FAC und FAC+1 kopieren können. Register sind dafür nicht geeignet, da wir sie dringender benötigen und ständig benutzt werden. Besser ist es, sich eine Speicherstelle zu schaffen.

Es gibt, wie Sie von Kurs II her sicher noch wissen, verschiedene Möglichkeiten, Speicherplatz zu reservieren. Wir benutzen hier DATA, genauso hätten wir auch BSS nehmen können.

Wir erweitern unser Programm wie folgt:

```
 .
 .
 .
       DEF ARITHM
       REF VSBW,NUMREF,XMLLNK
*
FAC    EQU >834A
CFI    EQU >1200
*
SCRNPO DATA >0000
*
ARITHM ...
       LI R0,0
       LI R1,1
       BLWP @NUMREF
       BLWP @XMLLNK
       DATA CFI
       MOV @FAC,@SCRNPO
 .
 .
 .
```

Unsere 142 befindet sich jetzt als >008E in der Speicherstelle SCRNPO und SCRNPO+1. Nehmen wir an, SCRNPO würde an Adresse >A100 beginnen, dann wären diese Adressen wie folgt benutzt:

```
>A100 >00
>A101 >8E
```

Jetzt können wir unseren numerischen Ausdruck einlesen. Dieser wird vor der Übergabe an das Maschinenprogramm ausgerechnet, so daß dort eigentlich auch wieder eine ganze Zahl vorliegt. Das Ergebnis aus 2*5-3 ist 7.

Überlegen Sie doch einmal, wie diese Zahl in der RADIX 100 SCHREIBWEISE dargestellt wird. Richtig?

```
>834A >40
>834B >07
>834C >00
>834D >00
>834E >00
>834F >00
>835A >00
>835B >00
```

Unser Exponent ist diesmal 0. Dies bedeutet, wir multiplizieren die Aneinanderreihung der übrigen Werte mit 100 hoch 0 (=1).

7 * 100 hoch 0 = 7.

Den Inhalt des Stapelspeichers können wir wieder mit XMLLNK umwandeln, wodurch unser Programm die folgende Form erhält. Der Wert >7 befindet sich danach in ZAHL.

```
 .
 .
       DEF ARITHM
       REF VSBW,NUMREF,XMLLNK
*
FAC    EQU >834A
CFI    EQU >1200
*
SCRNPO DATA >0000
ZAHL   DATA >0000
*
ARITHM ...
       LI R0,0
       LI R1,1
       BLWP @NUMREF
       BLWP @XMLLNK
       DATA CFI
       MOV @FAC,@SCRNPO
       LI R1,2
       BLWP @NUMREF
       BLWP @XMLLNK
       DATA CFI
       MOV @FAC,@ZAHL
 ...
```

Mit dieser ZAHL können wir nun in unserem Maschinenprogramm beliebig rechnen. Zum Beweis aber, daß sie richtig übergeben wurde, stellen wir sie auf dem Bildschirm dar. Dazu müssen wir aus dem Hexadezimalwert in ZAHL Dezimalziffern machen, die, als ASCII-Werte, auf dem Bildschirm darstellbar sind.

bei der Zahl 7 ist dies sehr einfach, wie bei allen einstelligen Zahlen. Wir addieren zum Hexadezimalwert in ZAHL einfach >30, und schon haben wir das ASCII-Zeichen. Das folgende Programm zeigt diese Schritte:

```
 .
 .
 .
       DEF ARITHM
       REF VSBW,NUMREF,XMLLNK
*
FAC    EQU >834A
CFI    EQU >1200
*
SCRNPO DATA >0000
ZAHL   DATA >0000
*
ARITHM ...
       LI R0,0
       LI R1,1
       BLWP @NUMREF
       BLWP @XMLLNK
       DATA CFI
       MOV @FAC,@SCRNPO
*
 .
 .
 .
```

```
.
.
.
        LI R1,2
        BLWP §NUMREF
        BLWP §XMLLNK
        DATA CFI
        MOV §FAC,§ZAHL
*
        MOV §SCRNPO,R0
        MOV §ZAHL,R1
        AI R1,>30
        SWPB R1
        BLWP §VSBW
.
.
.
```

Zuerst laden wir die Bildschirmposition, an der unsere Zahl dargestellt wird, in Register 0. Diese befindet sich ja gesichert in SCRNPO. Danach laden wir die Zahl in Register 1 und addieren den ASCII-Offset von >30. Der ASCII-Wert einer Ziffer ist immer genau >30 größer als der Ziffernwert, weshalb der Offset allgemeingültig ist.

Den so erhaltenen Wert wechseln wir ins höherwertige Byte und können dann die Zahl mittels VSBW darstellen.

Es kann nun aber durchaus passieren, daß die ZAHL mehr als eine Stelle hat. Für diesen Fall benötigt man eine Routine, die jede einzelne Stelle in einen ASCII-Wert umwandelt und die Werte beginnend bei der gegebenen Bildschirmposition ausgibt. Zu diesem Zweck ist es erforderlich, eine Hexadezimalzahl in eine dezimale umzuwandeln.

Nehmen wir also einmal an, der Wert in ZAHL sei 2574, also >0A0E.

```
>0A0E = >0*>10 >3 + >A*>10 2 + >0*>10 1 + >E*>10 0
      =  0 3   + 10 2  +  0 1  + 14 0
      =     0     +   2560   +     0    +   14
```

Jede Stelle muß mit einer 16er-Potenz multipliziert werden. Das Multiplizieren von 16 kann natürlich mit MPY, aber viel einfacher durch SLA Register,5 erreicht werden, denn eine fünfmalige Verschiebung nach links entspricht einer Multiplikation mit 16. Wenn Sie es nicht glauben, schauen Sie sich doch einfach nocheinmal unser Kapitel 1.2. an.

Mit einem entsprechendem Umwandlungsprogramm ist es jetzt möglich, jeden beliebigen Ausdruck auf den Bildschirm darzustellen (Offset beachten!), der als Ergebnis nicht kleiner als -32768 und nicht größer als 32767 ist.

Genauso ist es natürlich möglich, Variable zu übergeben. Die Sequenz im Basic,

```
10 A=142
20 B=2*5
30 C=-3
40 CALL LINK("ARITHM",A,B+C)
```

hat dieselbe Funktion, als wenn Sie für A direkt einen Wert einsetzen würden, ohne daß sich dadurch am Maschinenprogramm etwas ändert.

Um nun auch numerische Ordungen übergeben zu können, müssen Sie sich an die Erläuterungen in Verbindung mit den Stringutilities erinnern.

Die Elementenzahl wird genauso berechnet wie bei STRREF und STRASG und in Register 0 placiert.

Eine Beschreibung der Ordnung im LINK-Befehl erfolgt durch den Namen der Ordnung, die Klammern und eine Anzahl von Kommata, welche der Ausdehnung entspricht. Ein vierdimensionales Array wie Z(10,8,5,2) würde also wie folgt übergeben:

```
CALL LINK("START",Z(,,,)
```

Damit fehlt nur noch die Möglichkeit, numerische Parameter auch wieder an das Basic zurückgeben zu können. Wie dies funktioniert, zeigen die folgenden Seiten. Zunächst aber wieder ein paar Übungsaufgaben zu NUMREF und XMLLNK.

ÜBUNGSAUFGABEN

9. Welche der folgenden Aussagen sind falsch?

a) Um numerische Parameter zu übergeben, muß Register 2 die Parameterposition im LINK-Befehl enthalten.

b) Register 0 muß immer gleich Null sein, es sei denn, es wird eine Ordnung übergeben.

c) In Register 1 wird die Adresse angegeben, an der sich der übernommene oder der zu übergebende Wert befindet.

d) NUMREF ist in den Konsolenroutinen des Editor/Assemblers enthalten.

e) Die Größe der Zahlen, die übergeben werden können, ist beliebig.

f) Es können theoretisch Zahlen zwischen 8388607 und -8388608 übergeben werden.

g) Es können Zahlen von -32768 bis 32767 übergeben werden, ohne das besondere Software erforderlich ist.

h) Nach Durchführung von NUMREF befindet sich die Zahl, die übernommen wurde, im ASCII-Code an >834A und folgende.

i) XMLLNK ist in den BSCSUP-Utility-Routinen enthalten.

j) Das Programm zur Umwandlung eines Fließkommawertes in einen Festkommawert beginnt an >1200.

k) Die Startadresse des Umwandlungsprogramms wird XMLLNK in Register 0 übergeben.

l) FAC, CFI und CIF sind vordefinierte Symbole, die Sie nicht anders verwenden können.

m) Nach Durchführung der folgenden Routine befindet sich ein Wert, den wir mit CALL LINK("Name",x) übergeben haben, an Adresse >834A und >834B.

```
.
.
.
      LI R0,0
      LI R1,1
      BLWP §NUMREF
      BLWP §XMLLNK
      DATA >1200
.
.
.
```

n) Um XMLLNK verwenden zu können, muß es in einer REF-Anweisung am Programmanfang erwähnt werden.

o) Nach NUMREF können wir, genau wie bei STRREF, den eingelesenen Wert an einer bestimmten Adresse FAC belassen, um ihn später zu verwenden, auch, wenn wir weitere numerische Parameter übernehmen wollen.

p) Um eine Ziffer auf dem Bildschirm darzustellen, müssen wir den ASCII-Offset von >30 addieren.

10. Schreiben Sie ein Programm, welches die beiden integren numerischen Parameter einliest, diese miteinander multipliziert und das Ergebnis auf dem Bildschirm ausgibt. Folgendes Basicprogramm soll die Routine aufrufen:

```
10 CALL INIT
20 CALL LOAD("DSK1.MULTI")
30 INPUT A
40 IF A<0 THEN 30
50 IF A>1000 THEN 30
60 INPUT B
70 IF B<0 THEN 60
90 IF B>1000 THEN 60
90 CALL LINK("START",A,B)
```

11. Wie werden folgende Zahlen im Stapelspeicher FAC abgelegt, wenn Sie mit NUMREF einen numerischen Parameter übernehmen?

15, 111, 9041, 255, -18, -3099, 118.6, -255.9, 27.0003, 0.01314, -0.0001, 180781, 0, -255, 4096, 4097, 5000, -5000

12. Was befindet sich nach der Umwandlung mittels XMLLNK in den Adressen >834A und >834B (Zahlen wie Aufgabe 3)?

13. Schreiben Sie ein Programm, mit dessen Hilfe es möglich ist, einen String genau an einer Bildschirmkoordinate auf den Screen zu bringen. Die Koordinaten und der String sollen mit dem Basic Befehl CALL LINK(Zeile,Spalte,String) übergeben werden. Die Zeilen sind von 1 bis 24 und die Spalten von 1 bis 32 numeriert.

14. Schreiben Sie ein Maschinenprogramm, welches das Quadrat einer Zahl ermittelt. Aufruf mit LINK(Wert,Ergebnis).

15. Schreiben Sie ein Maschinenprogramm, welches die Basic-Funktion SEG$(String,Start,Ende) ersetzt. Aufruf mit LINK(String,Start,Ende,Teilstring-Ergebnis).

16. Schreiben Sie ein Maschinenprogramm, welches einen beliebigen Ausschnitt des Bildschirms, also ein Fenster, löschen kann. Der Aufruf aus dem Basic soll mit

CALL LINK("CLRWIN",Beginnzeile,-Spalte,Endzeile,-Spalte)

erfolgen, wobei diese Werte in dem unter 5) genannten Bereich rangieren.

17. Schreiben Sie ein Maschinenprogramm, welches es Ihnen erlaubt, durch CALL LINK("MODUS",1) und CALL LINK("MODUS",2) zwischen dem TEXT-Modus und dem GRAFIK-Modus hin- und herzuschalten. ACHTUNG: Das Aufrufende Basic-Programm darf nur enden oder unterbrochen werden, wenn Sie sich im GRAFIK-Modus befinden. Achten Sie einmal darauf, was bei PRINT, INPUT, DISPLAY etc. auf dem Bildschirm im TEXT-Modus geschieht!

Beschäftigen wir uns nun mit NUMASG. Mithilfe dieser Routine ist es möglich, numerische Parameter an das Basic zurückzugeben. Hierzu muß im LINK-Befehl des Basics an entsprechender Stelle eine numerische Variable stehen, also zum Beispiel:

```
CALL LINK("START",NUMV)
```

Um einen Wert zu übergeben, müssen wir diesen in unserem Stapelspeicher, der sich an Adresse >834A ff. befindet und den wir FAC genannt haben, aufbauen. Dazu schreiben wir die 16 Bit, die augenblicklich unseren Wert darstellen, in FAC und FAC+1. Diesen Wert wandeln wir anschließend mit dem 'Schwesterprogramm' von CFI, der XMLLNK-Utility-Routine CIF, um. CIF startet, wie berits erwähnt, an Adresse >2300.

Dazu schreiben wir ein Maschinenprogramm, welches den Wert 2055 ans Basic übergeben soll. Der Wert befindet sich in ZAHL.

```
 .
 .
 .
CIF     EQU >2300
ZAHL    DATA >0807
*
START   ...
        MOV @ZAHL,@FAC
        BLWP @XMLLNK
        DATA CIF
        BLWP @NUMASG
 .
 .
 .
```

Sie sehen, daß die Reihenfolge des Aufrufs von NUMASG und XMLLNK genau umgekehrt ist als beim Aufruf mit NUMREF. Selbstverständlich muß die zu schreibende Zahl ja erst umgewandelt, und dann übergeben werden.

Die Verwendung von DATA CIF setzt natürlich voraus, daß an irgendeiner Stelle im Programm, vor dem Durchlauf der Routine, CIF gleich zu >2300 gesetzt wurde. Allerdings funktioniert es ebenso mit DATA >2300.

Selbstverständlich können auch mit NUMASG wieder Ordnungen übergeben werden. Die Vorbereitung darauf erfolgt genauso, wie Sie es zusammen mit den String-Utilities und NUMREF gesehen haben.

.

Sie sehen bereits jetzt, daß durch die Verwendung der Parameter-Routinen die Verkettung von Basic und Maschinensprache eine interessante Sache wird.

Die Möglichkeiten, die sich daraus für Ihre eigenen Programm Entwicklungen ergeben, sind fast unbegrenzt.

Eine Reihe solcher Möglichkeiten finden Sie im Kapitel 3. Dort haben wir Auszüge unserer Programme MODE CONTROL und TORPEDO BASIC veröffentlicht, welche Maschinenroutinen für Basic zugänglich machen - wobei natürlich Parameter verwendet werden.

über das Thema Parameter könnte noch soviel gesagt werden, daß es den Rahmen dieses Buches sprengen würde. Wir wollen es daher bei dem belassen, was Sie bis jetzt wissen.

Bevor wir aber ganz zum Ende kommen, möchten wir uns noch kurz mit der ERR-Routine beschäftigen, welche ja ebenfalls im BSCSUP Paket enthalten ist.

Diese Routine erlaubt es nämlich, im Falle eines Fehlers, der im Maschinenprogramm auftritt, zum Fehlermeldungsprogramm des Basics zu verzweigen, den Fahler anzuzeigen und das Maschinenprogramm zu beenden, ohne daß das System abstürzt.

Jeder Fehler hat bekannterweise einen bestimmten Code. Diesen Code müssen Sie in das hochwertige Byte von Register 0 schreiben, und dann mit BLWP §ERR in die Fehlerroutine verzweigen.

Sie können damit zum Beispiel überprüfen, ob ein übernommener Parameter überhaupt im gültigen Bereich liegt - wie etwa eine Zeilennummer, die nur Werte zwischen 0 und 24 annehmen kann. Folgende Sequenz überprüft, ob FAC und FAC+1 in diesem Bereich liegen. Wenn nicht, wird ein BAD ARGUMENT Error ausgegeben und die Ausführung abgebrochen. Welche Fehlermeldungen möglich sind, zeigt eine Tabelle in Kapitel 5.

Doch nun zum Programm:

```
 .
 .
 .
       BL §BADARG
 .
 .
 .
BADARG MOV §FAC,R5
       CI R5,>01
       JLT OUT
       CI R5,>18
       JGT OUT
       JMP RETURN
OUT    LI R0,>1600
       BLWP §ERR
RETURN ...
 .
 .
 .
```

Auf diese Weise können Sie verhindern, daß falsche Werte ins Maschinenprogramm gelangen und das System abstürzt, wenn Sie aus dem Basic heraus eine MC-routine aufrufen.

In unserem Beispiel würde, wenn die übernommene Zahl kleiner als 1 oder größer als 24 ist, der Fehler

BAD ARGUMENT

ausgegeben, und die Programmkontrolle kehrt zum Basic zurück und unterbricht.

Im nächsten Kapitel werden Sie einige Besonderheiten erfahren, die es bei der Benutzung der Routinen mit dem EXTENDED BASIC gibt. Wer dieses Modul nicht besitzt, sollte sich mit dem dort vermittelten Wissen nicht belasten, da es ausschließlich Neues für die Besitzer dieses Moduls bringt.

Bevor Sie aber weiterlesen, wieder ein paar Übungsaufgaben zu den behandelten Utilities, wobei wir zunächst wieder nur Fragen über NUMASG und ERR, dann aber über das ganze Kapitel stellen. Schlagen Sie nicht in den Lösungen nach, sondern wiederholen Sie das Kapitel, wenn etwas unklar ist.

ÜBUNGSAUFGABEN

18. Schreiben Sie ein Programm, welches das kleinste gemeinsame Vielfache zweier Zahlen (0<n<100) ermittelt. Der Aufruf aus dem Basic soll der Art LINK(n1,n2,ermittelter Teiler) sein.

19. Wie lautet das Symbol der Routine, die eine Integerzahl in eine Fließkommazahl umwandelt und für XMLLNK von Bedeutung ist?

20. Wie können Sie Ordnungen an das Basicprogramm übergeben? Was ist dafür im Basic, was im Maschinenprogramm zu tun?

21. Nennen Sie 2 Möglichkeiten, zu verhindern, daß falsche Werte in das Maschinenprogramm gelangen.

22. Wo muß der Fehlercode an ERR übergeben werden?

23. Schreiben Sie ein Programm, welches den DISPLAY-Befehl des Extended Basic ersetzt. Folgende Parameter sollen übergeben werden:

1 = Druckzeile
2 = Druckspalte
3 = Stringlänge
4 = String

(Auf 'VALIDATE' wollen wir verzichten).

24. Welches der folgenden Programme übernimmt numerische Parameter vom Basic und gibt sie an ein Maschinenprogramm?

NUMASG, XMLLNK, NUMREF, CFI, CIF, STRASG, STRREF, ERR, VSBW?

25. Was bewirkt der folgende Programmteil?

```
 .
 .
 .
START  ...
       CLR R0
       LI R1,1
       BLWP §NUMREF
       BLWP §XMLLNK
       DATA >1200
       LI R0,>0000
       MOV §>834A,R3
       LI R1,>2000
S1     BLWP §VSBW
       INC R0
       DEC R3
       JNE S1
 .
 .
 .
```

26. Erläutern Sie folgende Begriffe:

- Integer;
- Fließkomma;
- Exponent;
- Radix 100 Schreibweise
- Stapelspeicher;
- Wertzuweisung;
- Parameter;
- Ordnung;
- 2er-Komplement;

27. Schreiben Sie ein Programm, welches die Summen der Inhalte aus zwei Ornungen voneinander subtrahiert. Die erste Ordnung soil dreidimensional, die zweite eindimensional sein. Die Wertzuweisung der Ordnungselemente soll durch eine Basic-Schleife erfolgen. Das Ergebnis steht in einer undimensionierten Variablen.

SOUND UTILITIES

Alle reden von den Grafikeigenschaften des TI-99/4a.

Daß dieser Computer auch hervorragende Soundmöglichkeiten hat, glaubt keiner! Mit diesem Programm,

SOUND UTILITIES

können Sie

- Sounds testen;

- Lieder Komponieren;

- Musiken speichern;

- diese in DATA-Zeilen umwandeln und

- in jedes Programm einbauen.

Ein LINE/LINE-Editor unterstützt Ihre Arbeit!

Tun Sie etwas für Ihre Ohren! Holen Sie sich die SOUND-UTILITIES ins Haus!!!

Diskette für XBasic/32Kb nur DM 39.90

1.10

1.10. MC-VERKETTUNG MIT EXTENDED BASIC

Vielleicht haben Sie es schon mehrfach versucht, aber dann aufgegeben, weil es nicht funktionierte: Die Verkettung von Maschinenprogrammen mit Extended Basic.

Obwohl im USER REFERENCE GUIDE ausdrücklich steht, daß MC-Programme auch auf dem Extended Basic Modul lauffähig sind, scheint das nicht so ohne weiteres zu funktionieren. Meißtens erscheint, wenn Sie versuchen, ein MC-Programm zu starten, das im TI-Basic mit E/A-Modul einwandfrei läuft, in Verbindung mit dem Extended Basic Modul nur die Fehlermeldung

```
* UNRESOLVED REFERENCE.
```

Unaufgelöster Bezug...! Dafür gibt es eigentlich nur eine logische Erklärung - daß nämlich irgendetwas an unseren so schön mit REF in die MC-Programme eingebundenen Utility-Startadressen nicht stimmt.

An REF VMBW,VSBW etc. scheint irgendetwas falsch zu sein.

Im Editor/Assembler-Modul sind diese Utilities an einer bestimmten Adresse, deren Eingangspunkt vordefiniert ist - eben mit VMBW, VSBW, VWTR und ähnlichen. Im Extended Basic Modul hingegen sind diese Eintrittsstellen nicht vordefiniert - wir müssen das selber tun.

Dies erreichen wir mit EQU. Auf der folgenden Seite finden Sie eine Tabelle mit den wichtigsten Gleichstellungen für Extended Basic; von den Utility-Routinen, die wir in diesem Kurs erwähnen. Eine komplette Übersicht über alle Utilities finden Sie im Handbuch zum Editor/Assembler und in Kapitel 5 dieses Buches.

VSBW
Video Single Byte Write
(Einzelne Bytes ins VDP-Ram schreiben)
Gleichstellung: VSBW EQU >2020

VMBW
Video Multi Byte Write
(Mehrere Bytes ins VDP-Ram schreiben)
Gleichstellung: VSBW EQU >2024

VSBR
Video Single Byte Read
(Einzelne Bytes vom VDP-Ram lesen)
Gleichstellung: VSBR EQU >2028

VMBR
Video Multi Byte Read
(Mehrere Bytes vom VDP-Ram lesen)
Gleichstellung: VMBR EQU >202C

VWTR
Video WriTe only Register
(Bytes in ein VDP-Register schreiben)
Gleichstellung: VWTR EQU >2030

KSCAN
Keyboard SCANner
(Tastaturüberwachung)
Gleichstellung: KSCAN EQU >201C

STRREF
STRing REFerence
(String-Parameter übernehmen)
Gleichstellung: STRREF EQU >2014

STRASG
STRing ASsinGn
(String-Parameter Zuweisung)
Gleichstellung: STRASG EQU >2010

NUMREF
NUMeric REFerence
(Numerische Parameter übernehmen)
Gleichstellung: NUMREF EQU >200C

NUMASG
NUMeric ASsiGn
(Numerische Parameter Zuweisung)
Gleichstellung: NUMASG EQU >2008

ERR
ERRor control
(Fehlerkontrolle)
Gleichstellung: ERR EQU > 2034

XMLLNK
eXtended Memory Language LiNKing
(Verkettung zu Utilityprogrammen im ROM)
Gleichstellung: XMLLNK EQU >2018

CFI
Carry Floating point to Integer
(Umwandlung von Fließkommazahlen in Integer)
Gleichstellung: CFI EQU >1288

CIF
Carry Integer to Floating point
(Umwandlung von Integerzahlen in Fließkomma)
Gleichstellung: CIF EQU >20

FAC
Floating point ACcumulator
(Fließkomma-Accumulator)
Gleichstellung: FAC EQU >834A

GPLWS
Graphik Programming Language WorkSpace
(Arbeitsbereich der Grafik-Programmiersprache)
Gleichstellung: GPLWS EQU >83E0

Wenn Sie ein MC-Programm mit Extended Basic laufen lassen wollen, ist es erforderlich, anstelle der REF-Bezüge die entsprechenden Gleichstellungen zu verwenden. Wir demonstrieren dies mit unserem Stringausgabe-Programm vom Anfang des vorangegangenen Kapitels.

```
       IDT  'STROUT'
       DEF  START
*
VMBW   EQU  >2024
STRREF EQU  >2014
*
GPLWS  EQU  >83E0
*
MYREG  BSS  >20
ADR    BSS  >FF
*
START  LWPI MYREG
       CLR  R0
       LI   R1,>0001
       LI   R2,ADR
       BLWP @STRREF
*
 .
 .
 .
```

Hier muß die Umwandlung des Strings erfolgen, welche den Offset berücksichtigt. Dieser Offset muß auch im Extended Basic bei der Bildschirmausgabe beachtet werden!

```
 .
 .
 .
       MOVB R0,@GPLSTA
       LWPI GPLWS
       B    @>0070
*
       END
```

Die Utility-Routinen werden also mit EQU und nicht mit REF bereitgestellt: können dann aber genauso benutzt werden, wie Sie es gewohnt sind. Der Aufruf erfolgt weiterhin mit BLWP @NAME.

Neben den Utility-Adressen gilt es zu beachten, daß Extended Basic auch Basic-Programme in die Speichererweiterung liest. Je nach Länge des Basicprogramms kann dieses beim Laden von Maschinenprogrammen überschrieben werden und dadurch zu Systemabstürzen kommen.

Sie können einen solchen Systemabsturz weitestgehend verhindern, wenn Sie für Ihr Maschinenprogramm eine Startadresse bestimmen, die vom Basic nicht so schnell benutzt wird. Dies erreichen Sie mit AORG.

AORG = Absolute ORiGin bestimmt die Startadresse für das Assemblerprogramm. Normalerweise sollten Sie am Anfang eines Maschinenprogramms, welches Sie mit Extended Basic laufen lassen wollen,

```
AORG >A000
```

spezifizieren. Diese Anweisung muß vor allen anderen Anweisungen stehen.

Die Unterschiede, wie die Speichererweiterung von Extended Basic und Editor/Assembler genutzt wird, zeigen zwei Tabellen in Kapitel 5.

Mehr Informationen zur Benutzung von Maschinenroutinen mit dem Extended Basic Modul finden Sie im Handbuch zum Editor/Assembler ab Seite 410.

Die Übungsaufgaben auf der folgenden Seite setzen voraus, daß Sie die entsprechenden Seiten im Handbuch zum Editor/Assembler und die Tabelle in Kapitel 5 dieses Buches über die Benutzung der Speichererweiterungseinheit gelesen haben.

ÜBUNGSAUFGABEN

1. Wieso funktionieren Maschinenprogramme, die mit dem TI-Basic des Editor/Assembler Moduls einwandfrei laufen, nicht auch ohne weiteres mit dem Extended Basic Modul?

2. Wie werden die Utility-Routinen Maschinenprogrammen zugänglich gemacht, die mit dem Extended Basic Modul laufen sollen?

3. Welche Ihnen bekannten Utility-Eintrittsstellen entsprechen bei der Benutzung des Extended Basic denen der nicht vordefinierten Symbole des Editor/Assemblers?

4. Wieso kann es zu Systemabstürzen kommen, wenn Sie die Startadresse des Maschinenprogramms nicht bestimmen?

5. Wie wird die Startadresse des Maschinenprogramms bestimmt?

6. Wo befinden sich die Utility-Routinen beim Extended Basic Modul; wo beim Editor/Assembler Modul?

7. Schreiben Sie die im letzten Kapitel in den Übungsaufgaben genannten Programme um, sodaß sie mit dem Extended Basic Modul lauffähig sind.

8. Schreiben Sie ein Programm, mit dessen Hilfe es möglich ist, einen Wert vom Basic direkt ins VDP-Ram zu schreiben (was mit dem CALL-Befehl nicht geht). Dieser Befehl ist im E/A bereits vorhanden und wäre sicher auch im Extended Basic nützlich.

1.11

1.11. ERSTELLEN DES SPEICHERAUSZUGS

In Kurs II haben wir dieses Thema bereits einmal erwähnt, sind aber nicht näher darauf eingegangen. Das wollen wir an dieser Stelle nachholen.

Wenn Sie mit dem Editor einen Assembler-Text eingeben, also die Labels, Mnemonics, Operanden und Kommentare in das Textsystem schreiben, so ist das noch kein Programm, mit dem der Computer etwas anzufangen weiß.

Alle Mnemonics und Operanden müssen erst in eine für den Computer lesbare Form gebracht werden. Diesen Vorgang nennt man assemblieren. Er wird ausgeführt, wenn Sie den ASSEMBLER geladen und Quell- und Objektcode-Dateinamen angegeben haben.

Nach Eingabe dieser beiden Dateinamen erscheinen noch die Fragen nach 'LIST FILE NAME' und 'OPTIONS'. Bei den Optionen war bisher nur 'R' für uns von Interesse, um den 'List File Name' haben wir uns noch gar nicht gekümmert.

Damit ist es nämlich möglich, herauszufinden, in welche Codes der Assembler die Mnemonics und Operanden umwandelt; an welcher Adresse sie sich befinden und wo unsere Programmeintrittstellen definiert sind.

Vielleicht haben Sie schon einmal eine reine Maschinenpprogramm-Datei gelistet (durch die PRINT-Option mit dem Objektcode) und mussten feststellen, daß diese Aneinanderreihung von Zahlen wenig übersichtlich ist.

Die LIST-Option des Assemblers hingegen gibt alle Informationen übersichtlich wieder:

a) Die Zeilennummer im Textsystem des Editors;
b) Die Adresse, an der sich der folgende Befehl befindet;
c) Die Codierung von Mnemonic und Operanden;
d) Der Programmtext in Reinschrift wie im Editor;
e) Die Kommentarzeilen

Bevor wir nun einmal überlegen, was sich mit einem solchen Speicherauszug anfangen lässt, wollen wir uns diesen einmal näher ansehen. Wir nehmen dazu einen Auszug aus unserem Programm PARTISAN VILLAGE, welches Ihnen von Kurs II her bekannt sein dürfte.

```
.
.
.
0129            *
0130            ***********************************************
0131            *                                             *
0132            *      INITIALISIERUNG DES SPIELS             *
0133            *                                             *
0134            ***********************************************
0135            *
0136            *      BILDSCHIRMAUFBAU
0137            *
0138 044E 1000  BEGIN1 NOP                  *SPIELSTART
0139 0450 04C0         CLR  R0              *R0 = Screen im VDP-Ram >0000
0140 0452 0201         LI   R1,SCRN1        *R1 = Startadresse Titelbild
     0454 0000'
0141 0456 0202         LI   R2,768          *768 Bildschirmpositonen
     0458 0300
0142 045A 0420         BLWP @VMBW           *an VDP-RAM uebergeben
     045C 0000
0143            *
0144 045E 0200         LI   R0,>0B00        *R0 = Sonderzeichen (ASCII 96)
     0460 0B00
0145 0462 0201         LI   R1,CHAR         *R1 = Startadresse Musterbesch
     0464 030C'
0146 0466 0202         LI   R2,216          *Bytes = Muster schreiben
     0468 00D8
```

```
 99/4 ASSEMBLER
VERSION 1.2                                                  PAGE 0007
      04C2 83CC
 0186 04C4 F820          SOCB @BYTE1,@>83FD     *VDP-Markierung, siehe
      04C6 044B'
      04C8 83FD
 0187                    *                      *Mnemonic-Erlaeuterungen SOCB
 0188 04CA D820          MOVB @BYTE1,@>83CE     *Musik starten
      04CC 044B'
      04CE 83CE
 0189 04D0 0300  TAST1B  LIMI 2                 *VDP-Interrupt-Umschaltung
      04D2 0002
 0190 04D4 0300          LIMI 0                 *VDP-Interrupt-Umschaltung
      04D6 0000
 0191 04D8 D820          MOVB @>83CE,@>83CE     *Ueberpruefen, ob alle Toene
      04DA 83CE
      04DC 83CE
 0192                    *                      *gespielt
 0193 04DE 13E7          JEQ  TAST1A            *Falls ja, von vorne beginnen
 0194 04E0 0300          LIMI 2
      04E2 0002
 0195 04E4 0300          LIMI 0
      04E6 0000
 0196                    *
 0197 04E8 0420          BLWP @K6CAN            *Tastaturabfrage
      04EA 0000
 0198 04EC D160          MOVB @>837C,R5         *GPL-Status-Byte in R5 schiebe
      04EE 837C
 0199 04F0 13EF          JEQ  TAST1B            *Wenn keine Taste Ruecksprung
 0200 04F2 0209          LI   R9,>0001          *Flag Taste gedrueckt
      04F4 0001
 0201 04F6 10EC          JMP  TAST1B            *Ruecksprung
 0202                    *                      *Ruecksprung zur Kontrolle
 0203                    *
 0204 04F8 04C9  LOS     CLR  R9                *Flag-Register Spielende loesc
 0205 04FA 1000  BEGIN3  NOP                    *s.o.
 0206 04FC 0208          LI   R8,>1FFF          *Verzoegerung laden
      04FE 1FFF
 0207 0500 0608  VERZ    DEC  R8                *vermindern
 0208 0502 16FE          JNE  VERZ              *Wenn nicht beendet Ruecksprun
 0209 0504 0205          LI   R5,>FF00          *Zeichen fuer Keine Taste lade
      0506 FF00
```


```
0210 0508 D805        MOVB R5,@>8375        *und uebertragen
     050A 8375
0211 050C 04C5        CLR  R5               *R5= >0000
0212 050E D805        MOVB R5,@>8374        *s.o.
     0510 8374
0213 0512 0420        BLWP @KSCAN           *s.o.
     0514 04EA'
0214              *                         *BEGINN DER UEBERUEFUNG:
0215 0516 04C5        CLR  R5               *R5 loeschen
0216 0518 D160        MOVB @>8375,R5        *Gegrueckte Taste in R5
     051A 8375
0217 051C 06C5        SWPB R5               *Bytes vertauschen
0218 051E 0285        CI   R5,>0045         *Ist Taste = <E>
     0520 0045
0219 0522 1300        JEQ  OBEN             *Wenn ja nach oben fahren
0220 0524 0285        CI   R5,>0044         *Ist Taste = <D>
     0526 0044
0221 0528 1317        JEQ  RECHTS           *Wenn ja nach rechts fahren
0222 052A 0285        CI   R5,>0058         *Ist Taste = <X>
```

```
 99/4 ASSEMBLER
VERSION 1.2                                                  PAGE 0013
  D BEGIN1  044E    ' BEGIN2  0496    ' BEGIN3  04FA    ' BYTE1   044B
   ' CHAR   030C    ' COLOR   03E4    ' CONTR   05A6    ' ENDALL  06C2
   ' ENDE   0626    ' ENDOUT  0678    ' HYMNE   068A    E KSCAN   0514
   ' LINKS  058C    ' LOOP0   0638    ' LOOP1   0648    ' LOOP2   0658
   ' LOS    04F8    ' NACHR   0300    ' NPAUS   06DC    ' NPAUS1  06D8
   ' OBEN   053E    ' ONE     03F2    ' PANZER  03FE    ' POS     044C
   ' POS1   03F6    ' PUNKTE  0400      R0      0000      R1      0001
     R10    000A      R11     000B      R12     000C      R13     000D
     R14    000E      R15     000F      R2      0002      R3      0003
     R4     0004      R5      0005      R6      0006      R7      0007
     R8     0008      R9      0009    ' RECHTS  0558    ' SCRN1   0000
   ' SETP   05C6    ' SIEG    05E6    ' SOUND   049E    ' SP      03F8
   ' SPALTE 03FA    ' SPENDE  06B6    ' TAST1A  04AE    ' TAST1B  04D0
   ' TOENE  0402    ' UNTEN   0572    ' VERZ    0500    ' VERZ1   06A4
   ' VERZ1A 06A0    E VMBR    0000    E VMBW    06D2    E VSBR    05AE
   E VSBW   0686    E VWTR    0484    ' ZEHN    03F4    ' ZEILE   03FC
   ' ZERO   03F0
  0000 ERRORS
```

Ganz am linken Rand finden Sie zunächst die Zeilennummer des Text-Editors. Diese wird mit ausgegeben und erlaubt so ein schnelles Wiederfinden der Zeilen.

Es folgt die jeweilige Adresse, welche das erste Byte der folgenden Befehlsgruppe enthält. Das erste Byte, manchmal auch die ersten zwei Bytes, entsprechen dem Mnemonic. Die folgenden Bytes bilden den oder die Operanden.

Jeder mnemotechnische Code hat einen bestimmten Wert, in den er durch den Assembler umcodiert wird. Nehmen wir einmal an, irgendwo stände der Befehl

```
LI R2,>A
```

Zu welcher Codierung würde dies führen?

LI (Load Immediate) wird zu >02.
Der erste Operand ist Register 2 und daher >02.
Der zweite Operand ist >A und daher >000A

Gehen wir nun einmal davon aus, daß sich der Ladebefehl an Adresse >06CC befindet, und im Editor in Zeile 368 geschrieben steht. In diesem Fall würde der Speicherauszug folgendes enthalten:

```
  .
  .
  .
0368 06CC 0202          LI   R2,>A       *Kommentar
     06CE 000A
  .
  .
  .
```

Dies bedeutet, daß im Speicher die Bytes wie folgt definiert sind:

```
>06CC >02 bzw.  00000010
>06CD >02 bzw.  00000010
>06CE >00 bzw.  00000000
>06CF >0A bzw.  00001010
```

Nehmen wir weiter an, der nun folgende Befehl lautet

BLWP §VMBW

Wie geht die Codierung nun weiter?

```
  .
  .
  .
0368 06CC 0202          LI   R2,>A       *Kommentar
     06CE 000A
0369 06D0 0420          BLWP §VMBW       *Kommentar
     06D2 069A
  .
  .
  .
```

Der Befehl startet bei >06D0. An dieser Adresse steht die Codierung für BLWP, welche >04 ist. Die folgenden beiden Bytes bestimmen die Adressierungsart; >02 steht für Adressierung durch ein Symbol. In >06D2 und >06D3 befindet sich die Adresse, zu der gesprungen werden soll. VMBW wird also in die Startadresse, die es symbolisiert, verwandelt.

In unserem Beispiel ist dies >069A. Damit sieht unser Speicher folgendermaßen aus:

```
>06CC >02 bzw.  00000010
>06CD >02 bzw.  00000010
>06CE >00 bzw.  00000000
>06CF >0A bzw.  00001010
>06D0 >04 bzw.  00000100
>06D1 >20 bzw.  00100000
>06D2 >06 bzw.  00000110
>06D3 >9A bzw.  10011010
```

Nehmen wir nun noch einen weiteren Befehl hinzu:

JNE THERE.

```
 .
 .
 .
0368 06CC 0202        LI   R2,>A       *Kommentar
     06CE 000A
0369 06D0 0420        BLWP @VMBW       *Kommentar
     06D2 069A
0370 06D4 16FA        JNE  THERE       *Kommentar
 .
 .
 .
```

Hier bestimmt das erste Byte an >06D4, daß es sich um einen bedingten Sprung JNE handelt, der als >16 codiert wird. Das nächste Byte gibt an, wie weit die Zieladresse, zu der gesprungen werden soll, von der augenblicklichen Adresse entfernt ist - und zwar in 2er-Komplementen-Form.

Dazu wieder unsere Byteliste:

>06CC >02 bzw. 00000010
>06CD >02 bzw. 00000010
>06CE >00 bzw. 00000000
>06CF >0A bzw. 00001010
>06D0 >04 bzw. 00000100
>06D1 >20 bzw. 00100000
>06D2 >06 bzw. 00000110
>06D3 >9A bzw. 10011010
>06D4 >16 bzw. 00010110
>06D5 >FA bzw. 11111010

Daraus ergibt sich, daß mit JUMP-Befehlen Sprünge maximal zu einer Adresse führen können, die 128 Bytepositionen entfernt ist. Wenn Sie größere Sprünge machen wollen, müssen Sie BRANCH verwenden. Andernfalls erhalten Sie beim Assemblieren die Fehlermeldung

* OUT OF RANGE.

Ein Rücksprung, bei dem in der Distanzbeschreibung das erste Bit gesetzt ist, führt im Programmablauf zurück. Ein Sprung, bei dem die Distanzbeschreibung mit einem Bit=0 beginnt, führt im Programm vorwärts. Der oben gezeigte Sprung führt also im Programm zurück.

Wir haben jetzt drei Befehle gezeigt, und jedesmal war die Codierung ein wenig verschieden. Es wäre nun aber sinnvoll zu wissen, wie die Codierung allgemein erfolgt. Dies wollen wir nun ausführlich tun. Eine Kurzübersicht finden Sie in den tabellen des Kapitels 5.

Natürlich ist es nun auch möglich, diese Tabellen umgekehrt einzusetzen. Sie können also aus der Codierung des Programms herausfinden, wie der Text im Editor lautet, und sich so eine bessere Vorstellung von der Arbeit des Programms machen. Diesen Vorgang nennt man 'Disassemblieren'. Ein Beispiel dazu bringen wir nach den folgenden Erläuterungen.

MNEMONIC-CODIERUNG

Jeder Assembler-Befehl, der im späteren Maschinenprogramm eine bestimmte Bedeutung hat, wird mit einem Byte codiert. Die folgende Tabelle zeigt dies in der Reihenfolge des Alphabets.

Mnemonic	Code	Format	Mnemonic	Code	Format
A	>A000	I	JOC	>1800	II
AB	>B000	II	JOP	>1C00	II
ABS	>0740	VI	LDCR	>3000	IV
AI	>0220	VIII	LI	>0200	VIII
ANDI	>0240	VIII	LIMI	>0300	VIII
B	>0440	VI	LREX	>03E0	VII
BL	>0680	VI	LWPI	>02E0	VIII
BLWP	>0400	VI	MOV	>C000	I
C	>8000	I	MOVB	>D000	I
CB	>9000	I	MPY	>3800	IX
CI	>0280	VIII	NEG	>0500	VI
CKOF	>03C0	VII	ORI	>0260	VIII
CKON	>03A0	VII	RSET	>0360	VII
CLR	>04C0	VI	RTWP	>03B0	VII
COC	>2000	III	S	>6000	I
CZC	>2400	III	SB	>7000	I
DEC	>0600	VI	SBO	>1D00	II
DECT	>0640	VI	SBZ	>1E00	II
DIV	>3C00	IX	SETO	>0700	VI
IDLE	>0340	VII	SLA	>0A00	V
INC	>0580	VI	SOC	>E000	I
INCT	>05C0	VI	SOCB	>F000	I
INV	>0540	VI	SRA	>0800	V
JEQ	>1300	II	SRC	>0B00	V
JGT	>1500	II	SRL	>0900	V
JH	>1B00	II	STCR	>3400	IV
JHE	>1400	II	STST	>02C0	VIII
JLE	>1200	II	STWP	>02A0	VIII
JLT	>1100	II	SWPB	>06C0	VI
JMP	>1000	II	SZC	>4000	I
JNC	>1700	II	SZCB	>5000	I
JNE	>1600	II	TB	>1F00	II
JNO	>1900	II	XOR	>2800	III

INFORMATIONEN

über unser reichhaltiges Programm an

- SOFTWARE
- HARDWARE
- ZUBEHÖR

für

- TI-99/4a
- SCHNEIDER CPC464/664

erhalten Sie Gratis bei Rausch & Haub!
Bitte unbedingt System angeben!

FORMATE

In der Tabelle finden Sie die Rubrik 'Format'. Diese bestimmt, was die folgenden Bytes enthalten.

Format I:
Im hochwertigen Nybble des zweiten Bytes befindet sich die Adressierungsart. BIN(00) bestimmt eine Operation mit einem Workspace-Register, BIN(10) symbolische Adressierung und BIN(11) bestimmt Workspace-Register-Benutzung mit automatischer Inkrementierung.
Die vier folgenden Bits enthalten das Workspace-Register für den Empangsoperanden.
Die nächsten beiden Bits enthalten die Adressierungsart für den Quelloperanden, wie oben gezeigt.
Abschließend enthalten die letzten vier Bits das Workspace-Register für den Quelloperanden.

Format II:
Die ersten 8 Bits enthalten die Mnemonic. In den zweiten acht Bits ist die Distanzstrecke zur Zieladresse in 2er-Komplement Form angegeben.

Format III:
Entsprechend Format I. Im hochwertigen Nybble des zweiten Bytes ist jedoch nicht die Adressierungsart für den Empfangsoperanden, sondern eine instruktionsspezifische Information untergebracht (Die Mnemonic besteht aus 2 Bytes).

Format IV:
In den ersten 5 Bits ist die Mnemonic enthalten. In den nächsten 4 Bits befindet sich die Anzahl der zu übertragenden Bits (Format IV betrifft CRU-Instruktionen; diese werden Sie noch kennenlernen), und die letzten 6 Bits entsprechen in ihrem Format dem Format 1.

Format V:
In den ersten 8 Bits befindet sich die Mnemonic. Die folgenden 4 Bits enthalten den Zähler, und die restlichen Bits die Workspace-Registeradresse. Format V betrifft die Workspace-Register-Verschiebeinstruktionen.

Format VI:
In diesem Format enthalten die letzten 6 Bits dieselben Informationen, wie bei Format 1 gezeigt. Die übrigen Bits enthalten spezifische Daten.

Format VII:
Die letzten fünf Bits enthalten Nullen.

Format VIII:
Die letzten vier Bits enthalten eine Workspace-Register Adresse.

Format IX:
Siehe Format III.

Format X:
Die letzten vier Bits enthalten eine Workspace-Register Adresse.

Eine Übersicht, wie die einzelnen Formate aufgebaut sind, finden Sie in Kapitel 5.

OPERANDEN

Sind für die Durchführung des Befehls weitere Angaben erforderlich, so sind diese in den folgenden Bytes enthalten, und zwar in folgender Form:

Symbole: Angabe der Adresse, die sie symbolisieren.

Direktdaten: In Reinschrift, also >FFFF als >FFFF etc.

Dies gilt auch bei indirekter Adressierung.

Am Ende des Speicherauszugs gibt der Assembler alle Symbole mit ihren Adressen in alphabetischer Reihenfolge an. Wenn R als Option gewählt wurde, werden auch die Register in dieser Liste mit ihrer Adresse angegeben.

Vor jedem Symbol kann ein Zeichen stehen, welches Aufschluß über die Bedeutung gibt.

D - bezeichnet ein in einer DEF-Instruktion genanntes Symbol.

R - bezeichnet einen Externen Bezug durch REF.

' - bezeichnet sonstige Eintrittsstellen in ein Programm, die nicht mit DEF oder REF spezifiziert wurden.

Hinter E werden zum Beispiel auch die eingeschlossenen Utility-Routinen (VSBW, VMBW etc.) genannt.

Mit diesem Wissen können Sie ein Programm Disassemblieren, das heißt, vom Objektcode in den Quellencode zurückübersetzen.

Beispiel:

```
 .
 .
 .
05F8 >C8 bzw.  11001000
05F9 >00 bzw.  00000000
05FA >04 bzw.  00000100
05FB >4C bzw.  01001100
 .
 .
*
 .
 .
*    POS     044C
```

>C... steht für den MOV-Befehl. Dieser hat das Format I. Nach der Beschreibung dieses Formates enthalten die folgenden Bytes:

>C8.. (8=1000) für eine Wortoperation mit Workspace-Register Adressierung.

>C800 (00=0000) gibt das Workspace-Register des Quelloperanden an, in diesem Fall R0.

>044C ist der zweite Operand, nämlich das im Speicherauszug genannte Symbol, welches mit POS gekennzeichnet ist.

Damit haben wir den kompletten Befehl. Er lautet:

```
MOV R0,@POS
```

Direktiven, die vom Assembler erkannt werden, fallen natürlich nicht unter die gezeigte Codierung, denn sie tauchen später im Programm nicht mehr auf.

Folgendes wird von der LIST Option ausgegeben, wenn Sie Direktiven in Ihrem Programm verwenden:

'TEXT'
a) Editorzeile
b) Speicherstelle des ersten Bytes der Textzeile
c) ASCII-Wert des ersten Zeichens der Textzeile
d) Programm-Text und Kommentare

'DATA'
a) Editorzeile
b) Speicherstelle des jeweiligen 2-Byte-Worts
c) Inhalt des 2-Byte-Wortes
d) Programm-Text und Kommentare

'BYTE'
a) Editorzeile
b) Speicherstelle des Byte-Ausdrucks
c) Inhalt des Bytes (Wert)
d) Programmtext und Kommentare

DEF, REF, EVEN, BSS, BES führen zu keiner Ausgabe in der Mnemonic- und Operandenspalte, es wird aber ggf., so bei BSS, die Adresse für nachfolgende Befehle erhöht.

Sie werden sich nun warscheinlich fragen, was es bringt, zu wissen, wo was im Speicher enthalten ist. Auf den folgenden Seiten wollen wir Ihnen dieses sagen.

SYMBOL TABLE OVERFLOW - was tun?

Wenn Sie zuviele Eingangsstellen in Ihrem Programm angeben, kann es passieren, daß der Assembler keinen Platz mehr findet, diese alle zu registrieren.

Da Sie jetzt wissen, wo sich was im Speicher befindet, können Sie aber Symbole streichen und diese direkt durch entsprechende Adressenangaben ersetzen.

Ziehen wir dabei noch einmal unser Disassemblierungsbeispiel heran. Der Befehl, den wir aus dem Code herausgefunden haben, lautete:

MOV R0,@POS

Aus unserer Symbol-Tabelle wissen wir, daß POS an Adresse >044C beginnt. Wir können nun zum Beispiel auch schreiben:

MOV R0,@>044C

Damit sind wir auf das Symbol POS nicht mehr angewiesen und können es als Eintrittsstelle streichen.

Solche Adressenbezeichnungen können wir bei den meißten Befehlen verwenden. Zum Beispiel bei JMP. Nehmen wir an, ADR sei >0620 und wir treffen auf den Befehl JMP ADR. Genausogut können wir nun Schreiben:

JMP >0620

Wenn wir bei einem Sprung von der augenblicklichen Position ausgehen möchten, können wir das Dollar-Zeichen verwenden. Dieses beschreibt die aktuelle Position des Programmzeigers. Beispiel:

JMP $+4

überspringt die nächsten 4 Bytes im Programm. Wir brauchen also nicht für jede Alternativ-Verzweigung eine neue Eingangsstelle, sondern müssen lediglich wissen, wieviele Bytes durch die zu überspringenden Daten und Befehle in Anspruch genommen werden.

DATA-ZEILEN

Bestimmt wollten Sie mit einem/einer Bekannten schon einmal selbstgeschriebene Assembler-Programme tauschen. Leider scheiterte dies bislang daran, daß Ihr(e) Freund(in) keinen Assembler besitzt und den Quellcode nicht abtippen konnte. Jetzt besteht die Möglichkeit, die Speicherinhalte in DATA-Zeilen zu verwandeln und mittels LOAD in die Speichererweiterung zu poken.

Ein so gepoktes Programm läuft genauso, als wäre es mit dem Editor erstellt worden. Es ist nur sehr mühsam, solche Data-Zeilen zu entwickeln - wenn man nicht selbst das ganze als Speicherauszug ausgelistet bekommt. Aber natürlich ist es auch möglich, ein MC-Programm 'per Hand' zu assemblieren.

Nur eines gilt es zu beachten:

Wenn Sie die Daten in die Speichererweiterung poken, wird noch keine Eintrittstelle für den Aufruf aus dem Basic generiert. Diese müssen Sie separat in der REF/DEF-Tabelle unterbringen. Wie das funktioniert, steht im Handbuch zum Editor/Assembler und ist eigentlich ganz einfach, obwohl eine genaue Erläuterung den Rahmen dieses Buches sprengen würde.

Wenn Sie also zum Beispiel auf den gezeigten MOV-Befehl treffen, können Sie im Basic (dort natürlich eleganter durch eine Schleife, denn es ist ja nicht nur ein Befehl zu poken) schreiben:

```
 .
 .
 .
130 CALL LOAD(1528,200,0,4,76)
 .
 .
 .
```

Beachten Sie auch, daß der Maschinenprogrammstart festgelegt werden sollte, um nicht mit dem Basicprogramm 'zusammenzuecken'.

Allein diese beiden Anwendungen zeigen, daß es schon sehr sinnvoll ist zu wissen, wo was im Speicher passiert, wenn ein Maschinenprogramm läuft. Es lässt sich sehr viel damit anfangen, wenn man nicht mehr unbedingt auf die Symbole angewiesen ist.

Nehmen Sie sich also am Besten einmal ein paar Ihrer selbstentwickelten Programme vor, und drucken Sie einen Speicherauszug aus. Sehr warscheinlich werden Sie die ein oder andere Stelle finden, die Sie besser Programmieren können. Hin und wieder lassen sich dadurch sogar ein paar Bytes Speicherplatz sparen.

Auf den nachfolgenden Seiten nun wieder ein paar Übungsaufgaben..

ÜBUNGSAUFGABEN

1. Welche Schritte sind erforderlich, um einen Speicherauszug zu erhalten?

2. Wie erfolgt die Ausgabe des Speicherauszugs auf den Drucker?

3. Assemblieren Sie folgendes Programm 'per Hand':

```
        DEF  START
        REF  VSBW
*
START   LI   R0,>0000
        LI   R1,>2000
        BLWP @VSBW
        INC  R0
        JNE  $-6
        RT
```

4. Disassemblieren Sie folgenden Programmteil 'per Hand':

```
  .
  .
  .
>0496  >10
>0497  >00
 0498  >04
>0499  >C5
 049A  >DB
 049B  >05
 049C  >83
 049D  >74
  .
  .
  .
```

5. Disassemblieren Sie folgenden Programmteil 'per Hand':

```
 .
 .
 .
>04FE  >1F
>04FF  >FF
>0500  >06
>0501  >08
>0502  >16
>0503  >FE
>0504  >02
>0505  >05
>0506  >FF
>0507  >00
 .
 .
 .
```

6. Beschreiben Sie die Zusammensetzung eines Maschinenbefehls im Format I, im Format II und im Format V!

7. Worin unterscheiden sich die Formate III und IV?

8. Welche Auswirkungen haben die folgenden Direktiven bei der Ausgabe der Speicherauszugsliste?

- REF, DEF, BYTE, TEXT, UNL.

9. Was wird durch das Dollar-Zeichen symbolisiert?

10. Worauf muß geachtet werden, wenn mit LOAD Daten als Maschinen-Programm in die Speichererweiterung gepoked werden sollen?

BITEMAN

Biteman von HAGERA(R) ist ein kleines Männchen voller Tatendrang und immer auf der Suche nach Apfelsinen und Kraftpillen. Doch nur Listigkeit kann ihn vor dem fast unverwundbaren Hausgeist retten...

Die Handlung dürfte vielen nicht unbekannt sein. BITEMAN ist ein amüsantes Spiel für Junge und Junggebliebene.

BITEMAN gibt es für TI-99/4a, XB + Joysticks

auf CASSETTE oder DISKETTE für ganze DM 19,90

HEUTE NOCH BESTELLEN!

1.12

1.12. DIE CRU-INSTRUKTIONEN

Mit einer Befehlsgruppe haben wir uns bisher noch gar nicht beschäftigt: Mit den CRU-Instruktionen. Wenn dieses Kapitel auch nicht alle Fragen beantworten kann (nähere Informationen finden Sie in Ihrem Handbuch zum Editor/Assembler), so wollen wir doch die Grundlagen vermitteln.

Was bedeutet eigentlich CRU?

CRU steht für Communication Register Unit, also auf deutsch soviel wie 'Kommunikations-Register-Einheit). Es handelt sich dabei um einen 4 Kilobyte großen Bereich, der dazu benutzt wird, Systemzugänge zu Peripheriegeräten zu haben. Der Bereich wird durch >0000 bis >1FFF gekennzeichnet, hat also im Prinzip die gleiche 'Adress-Codierung' wie schon unser Ram und das VDP-Ram.

>0000 bis >07FE wird von durch das Konsolen-Rom benutzt. Durch das Herz des Systems, den TMS 9900 I/O-Chip, werden zum Beispiel die Tastatur und der Cassetteninput/output angesteuert.

>0800 bis >0FFE ist für die spätere Verwendung reserviert. Da Texas Instruments jedoch die Produktion eingestellt hat, ist fraglich, ob dieser Bereich überhaupt noch jemals genutzt wird.

>1000 bis >1FFE ist für die Cardridges und Module reserviert, mit denen die TI-Console bestückt werden kann. Jedem Peripherie-Gerät wird dabei ein Block mit 128 Bytes zugeteilt. Sie finden eine Tabelle im Handbuch zum Editor/Assembler auf Seite 407.

Die Benutzung von Peripherien - dazu gehören auch die Tastatur, die Joysticks und die Cassettenroutine, ist Interruptgesteuert. Mittels der CRU ist es möglich, Zugriff auf diese Steuerung zu nehmen. Welche Interrupts möglich sind, zeigt eine weitere Tabelle im E/A-Handbuch.

Ausgegangen wird bei der Benutzung der CRU von der sogenannten CRU-Basis Adresse.

Die CRU-Basis-Adresse befindet sich in Workspace-Register 12. Sie ist um eine Bitstelle im Register nach links geschoben und daher in den Bits 3 bis 14 enthalten. Die folgenden fünf Befehle sind für die Benutzung der CRU-Sektion wesentlich:

LDCR	LoaD CRu	CRU laden
STCR	STore CRu	CRU speichern
SBO	Set cru Bit to One	CRU Bit auf 1 setzen
SBZ	Set cru Bit to Zero	CRU Bit auf 0 setzen
TB	Test Bit	CRU Bit prüfen

Die Funktionsweise dieser Befehle ist ausführlich im Handbuch zum Editor/Assembler erläutert. Wir wollen dies deshalb in diesem Kurs nicht wiederholen, zumal der fortgeschrittene Programmierer (zu denen Sie sich nach diesem Kurs durchaus zählen können), sicher auch selbst gerne noch etwas neues an seinem Computer entdeckt.

Neben den genannten Befehlen gibt es einige weitere, ver deren Benutzung mit dem Homecomputer im Handbuch jedoch ausdrücklich gewarnt wird, da diese zur Verwendung mit anderen Geräten, welche denselben Microprozessor enthalten, bestimmt sind. Es handelt sich dabei um Befehle zur Steuerung der Zeittakt-Uhr.

CKON	ClocK ON	Interruptuhr ein
CKOF	ClocK OFf	Interruptuhr aus
IDLE	IDLE	Übersprung
RSET	ReSET	Zurücksetzen
LREX	Load or Restart EXecute	Laden oder Neustart

Der Assembler des E/A-Pakets erkennt diese Befehle und sie werden auch im Programm ausgeführt.

Wir wollen es dabei bewenden lassen. Professionelle Anwender (und nur für solche sollten CRU-Manipulationen von Interesse sein) werden sicher keine Schwierigkeiten bei der Benutzung der CRU-Befehle haben.

ÜBUNGSAUFGABEN

1. Was versteht man unter CRU und welche Bedeutung haben die einzelnen CRU-Befehle LDCR, STCR, SBO, SBZ und TB?

2. Wie können Sie die CRU-Basis-Adresse ermitteln?

SCHLUSSWORT

Damit wären Sie am Ende dieses Kurses angelangt. Wir hoffen, daß Sie nach der Lektüre der Kapitel ein wenig klarer sehen in der komplizierten Materie der Assembler-Programmierung.

Sollten Sie beim Lernen auf Hindernisse stoßen, so empfehlen wir Ihnen unseren Einsteigerkurs 'ASSEMBLER KURS II', welcher ebenfalls direkt über uns zu beziehen ist. Einen entsprechenden Hinweis finden Sie am Anfang dieses Buches.

Im nun folgenden Kapitel finden Sie die Lösungen zu den Übungsaufgaben des Kurses. Schlagen Sie dort aber nur nach, wenn Sie anders nicht mehr weiterkommen. Zu einigen Aufgaben sind auch keine direkten Lösungen angegeben - wenn sich diese mit in der Kurslektion oder im Softwareteil befindet. In diesen Fällen haben wir besonders darauf hingewiesen.

Wir wollen Ihnen auch einen abschließenden 'Test' zum Gelernten ersparen und lieber empfehlen, die Kapitel mehrmals zu lesen, um mehr Programmiersicherheit zu erhalten. Wir weisen jedoch darauf hin, daß nur die Praxis Ihre Programmiererfahrung erhöhen kann. Arbeiten Sie also am besten direkt am Gerät mit den neuvermittelten Kenntnissen.

Über Anregungen, Verbesserungsvorschläge und Kritik zu diesem Buch würden wir uns sehr freuen. Bei Fragen schreiben Sie bitte unter Hinzufügung eines frankierten und adressierten Freiumschlages. Unsere Adresse finden Sie eigentlich überall und am Anfang dieses Buches.

BELOHNUNG

Wir PLANEN:

ASSEMBLER KURS I (Grundlagen)
ASSEMBLER KURS IV (für Profis)
ASSEMBLER KURS V (Praxis)

Für die Assembler Kurse I+IV suchen wir Erfahrungsberichte im Umgang mit

- Binär-Logik
- TI-Speicher-Manipulationen
- Externen Dateien
- CRU-Manipulationen
- Interrupt-Steuerung

Jeder Einsender, dessen Bericht abgedruckt wird, erhält nach Fertigstellung des Buches (ca. Ende 1986) ein Freiexemplar.

Für den Assembler Kurs V suchen wir erstklassige, kommentierte Programme in Assembler, insbesondere:

- Textbearbeitung
- Dateiverwaltung
- rasante Spiele
- Utilities

Jeder Einsender, dessen Programm im Buch veröffentlicht wird, erhält ein zu vereinbarenendes Honorar (100-300 DM) und ein Freiexemplar.

PREISAUSSCHREIBEN

Unter allen Einsendern, deren Artikel, Berichte, Texte oder Programme in einem der genannten Bücher veröffentlicht werden, ermitteln wir den besten Beitrag.

==========

Dieser Beitrag wird mit DM 1000.- honoriert!

==========

Teilnahmebedingungen:

Für Texte aller Art ist ein sauberes Manuskript, möglichst mit einem Textverarbeitungsprogramm geschrieben, einzusenden. Der Zeilenabstand soll mindestens 2zeilig sein, rechter Rand mindestens 6 cm.

Programm-Einsendungen nur auf Diskette, lauffähig auf TI-99/4a mit Speichererweiterung, Extended Basic oder Editor Assembler Modul. Die Programme müssen sowohl im Objekt als auch im Quellencode auf der Diskette enthalten sein. Dazu bitte eine ausführliche Dokumentation (wie oben beschrieben), was das Programm macht und eine separate Bedienungsanleitung.

Wichtig für alle Einsendungen:

1. Alle Einsendungen bitte mit Honorarvor- stellung.

2. Freiumschlag für eine evtl. Rücksendung beilegen!

3. Der Rechtsweg bei der Teilnahme am Preisausschreiben ist ausgeschlossen.

4. Einsendeschluss für die Teilnahme am Preisausschreiben ist der 30.06.1986!

5. Das Honorar wird 2 Monate nach Drucklegung ausgezahlt. Bei Annahme eines Programmes oder Textes wird ein Autorenvertrag abgeschlossen.

6. Vergabe des Sonderpreises, der zusätzlich zu einem vereinbarten Honorar gezahl wird, ist Anfang Dezember 1985. Die Ermittlung des Preisträgers erfolgt durch RAUSCH & HAUB unter Ausschluß des Rechtsweges. Die Form der Ermittlung ist RAUSCH & HAUB vorbehalten.

7. RAUSCH & HAUB behält sich ferner vor, keine Verträge abzuschließen, den Sonderpreis zurückzuhalten oder auf meherere Personen zu verteilen, wenn keines oder mehrere Beiträge den von uns gestellten Anforderungen entsprechen.

Und jetzt ran an den TI und viel Spaß!!!

2

LÖSUNGEN ZU DEN ÜBUNGSAUFGABEN

In diesem Kapitel finden Sie die Lösungen zu den meißten in diesem Buch gestellten Übungsaufgaben. Die gezeigten Lösungen müssen, insbesondere bei Problemstellungen, nicht immer den besten oder schnellsten Weg darstellen. Wir überlassen es Ihnen, günstigere Lösungswege zu finden.

Bei einigen Aufgaben haben wir auf die Angabe eines kompletten Lösungsweges verzichtet oder nur das Lösungsprinzip angegeben, da der Abdruck einer komplexen Lösung den Rahmen dieses Buches beiweitem sprengen würde. Auch verweisen wir in diesem Zusammenhang auf unsere DISKETTENEMPFEHLUNGEN.

Sollten bei der Lösung der ein oder anderen Aufgabe größere Schwierigkeiten auftreten, so schreiben Sie uns bitte. Wir sind bemüht, diesen Kurs für Sie als fortgeschrittenen Assembler-Programmierer so verständlich wie möglich zu halten.

Für Fragen, welche aufgrund mangelndem Grundwissen auftreten, verweisen wir jedoch nochmals auf unseren Einsteigerkurs ASSEMBLER KURS II. Die Beantwortung derartiger Fragen obliegt nicht dem Zweckbereich dieses Bandes.

LÖSUNGEN ZU KAPITEL 1.1.

1. Binärdarstellung

9	0000 0000 0000 1001
1056	0000 0100 0020 0000
220	0000 0000 1101 1100
-7	1111 1111 1111 1001
118	0000 0000 0111 0110
513	0000 0010 0000 0001
-512	1111 1110 0000 0000
-1	1111 1111 1111 1111
900	0000 0011 1000 0100
256	0000 0001 0000 0000
-210	1111 1111 0010 1110

2. ANDI und ORI

9	0000 0000 0000 1001
4299 (>10CB)	0001 0000 1100 1011
ANDI	0000 0000 0000 1001
ORI	0001 0000 1100 1011
1056	0000 0100 0010 0000
4299 (>10CB)	0001 0000 1100 1011
ANDI	0000 0000 0000 0000
ORI	0001 0100 1110 1011
220	0000 0000 1101 1100
4299 (>10CB)	0001 0000 1100 1011
ANDI	0000 0000 1100 1000
ORI	0001 0000 1101 1111

Fortsetzung zu Aufgabe 2.

-7	1111 1111 1111 1001
4299 (>10CB)	0001 0000 1100 1011
---	---
ANDI	0001 0000 1100 1001
ORI	1111 1111 1111 1011
118	0000 0000 0111 0110
4299 (>10CB)	0001 0000 1100 1011
---	---
ANDI	0000 0000 0100 0010
ORI	0001 0000 1111 1111
513	0000 0010 0000 0001
4299	0001 0000 1100 1011
---	---
ANDI	0000 0000 0000 0001
ORI	0001 0010 1100 1011
-512	1111 1110 0000 0000
4299	0001 0000 1100 1011
---	---
ANDI	0001 0000 0000 0000
ORI	1111 1110 1100 1011
900	0000 0011 1000 0100
4299	0001 0000 1100 1011
---	---
ANDI	0000 0000 1000 0000
ORI	0001 0011 1100 1111
256	0000 0001 0000 0000
4299	0001 0000 1100 1011
---	---
ANDI	0000 0000 0000 0000
ORI	0001 0001 1100 1011

Fortsetzung zu Aufgabe 2.

```
 -210              1111 1111 0010 1110
 4299              0001 0000 1100 1011
-------------------------------------
ANDI               0001 0000 0000 1010
ORI                1111 1111 1110 1111
```

1. XOR kann überhaupt keine Direktdaten verknüpfen. Daher erhalten Sie als Ergebnis lediglich eine Fehlermeldung, wenn Sie versuchen, das Programm zu assemblieren. Der zu verknüpfende Wert muß vorher an eine Adresse geladen werden.

```
LI R6,>10CB
LI R1,9
MOV R1,@ADR
XOR @ADR,R6
```

Jetzt können Sie XOR mit ADR durchführen, was mit den Werten aus Aufgabe 2 zu folgendem Ergebnis führen würde:

```
    9              0000 0000 0000 1001
 4299 (>10CB)      0001 0000 1100 1011
-------------------------------------
XOR                0001 0000 1100 0010

 1056              0000 0100 0010 0000
 4299 (>10CB)      0001 0000 1100 1011
-------------------------------------
XOR                0001 0100 1110 1011

  220              0000 0000 1101 1100
 4299 (>10CB)      0001 0000 1100 1011
-------------------------------------
XOR                0001 0000 0001 0111

   -7              1111 1111 1111 1001
 4299 (>10CB)      0001 0000 1100 1011
-------------------------------------
XOR                1110 1111 0011 0010
```

Fortsetzung zu Aufgabe 3.

```
  118              0000 0000 0111 0110
 4299 (>10CB)      0001 0000 1100 1011
-----------------------------------------
XOR                0001 0000 1011 1101

  513              0000 0010 0000 0001
 4299              0001 0000 1100 1011
-----------------------------------------
XOR                0001 0010 1100 1010

 -512              1111 1110 0000 0000
 4299              0001 0000 1100 1011
-----------------------------------------
XOR                1110 1110 1100 1011

  900              0000 0011 1000 0100
 4299              0001 0000 1100 1011
-----------------------------------------
XOR                0001 0011 0100 1111

  256              0000 0001 0000 0000
 4299              0001 0000 1100 1011
-----------------------------------------
XOR                0001 0001 1100 1011

 -210              1111 1111 0010 1110
 4299              0001 0000 1100 1011
-----------------------------------------
XOR                1110 1111 1110 0101
```

4. Dieses Problem wird mit SOCB gelöst. Hier ein Beispiel:

```
Quelle      0101 0101 0111 0101
Ziel        1001 0010 1001 1011
SOCB        1101 0111 1001 1011
```

5. DOC, SOCB, SZC und SZCB

9	0000 0000 0000 1001
4299 (>10CB)	0001 0000 1100 1011
---	---
SOC	0001 0000 1100 1011
SOCB	0001 0000 1100 1011
SZC	0001 0000 1100 0010
SZCB	0001 0000 1100 1011
1056	0000 0100 0010 0000
4299 (>10CB)	0001 0000 1100 1011
---	---
SOC	0001 0000 1110 1011
SOCB	0001 0100 1100 1011
SZC	0001 0000 1100 1011
SZCB	0001 0000 1100 1011
220	0000 0000 1101 1100
4299 (>10CB)	0001 0000 1100 1011
---	---
SOC	0001 0000 1101 1111
SOCB	0001 0000 1100 1011
SZC	0001 0000 0000 0011
SZCB	0001 0000 1100 1011
-7	1111 1111 1111 1001
4299 (>10CB)	0001 0000 1100 1011
---	---
SOC	1111 1111 1111 1011
SOCB	1111 1111 1100 1011
SZC	0000 0000 0000 0010
SZCB	0000 0000 1100 1011
118	0000 0000 0111 0110
4299 (>10CB)	0001 0000 1100 1011
---	---
SOC	0001 0000 1111 1111
SOCB	0001 0000 1100 1011
SZC	0001 0000 1000 1001
SZCB	0001 0000 1100 1011

Fortsetzung zu Aufgabe 5.

```
  513              0000 0010 0000 0001
 4299              0001 0000 1100 1011
-----------------------------------
SOC                0001 0010 1100 1011
SOCB               0001 0010 1100 1011
SZC                0000 0000 1100 1010
SZCB               0001 0000 1100 1011

 -512              1111 1110 0000 0000
 4299              0001 0000 1100 1011
-----------------------------------
SOC                1111 1110 1100 1011
SOCB               1111 1110 1100 1011
SZC                0000 0000 1100 1011
SZCB               0000 0000 1100 1011

  900              0000 0011 1000 0100
 4299              0001 0000 1100 1011
-----------------------------------
SOC                0001 0011 1100 1111
SOCB               0001 0011 1100 1011
SZC                0001 0000 0100 1011
SZCB               0001 0000 1100 1011

  256              0000 0001 0000 0000
 4299              0001 0000 1100 1011
-----------------------------------
SOC                0001 0001 1100 1011
SOCB               0001 0001 1100 1011
SZC                0001 0000 1100 1011
SZCB               0001 0000 1100 1011

 -210              1111 1111 0010 1110
 4299              0001 0000 1100 1011
-----------------------------------
SOC                1111 1111 1110 1111
SOCB               1111 1111 1100 1011
SZC                0000 0000 1100 0001
SZCB               0000 0000 1100 1011
```

LÖSUNGEN ZU KAPITEL 1.2

1. Indirekte Verschiebung mit SRL, SLA und SRC.

Inhalt von R0: 0100 0110 1110 0101
Inhalt von R5: 1011 1101 0111 1010

SLA R5,0

1011 1101 0111 1010
0111 1010 1111 0101
1111 0101 1110 1011
1110 1011 1101 0111
1101 0111 1010 1111

Der neue Wert von Register 5 lautet 1101 0111 1010 1111.

SRA R5,0

1011 1101 0111 1010
1101 1110 1011 1101
1110 1111 0101 1110
1111 0111 1010 1111
1111 1011 1101 0111

Der neue Wert von Register 5 lautet 1111 1011 1101 0111.

SRC R5,0

1011 1101 0111 1010
0101 1110 1011 1101
1010 1111 0101 1110
0101 0111 1010 1111
1011 1011 1101 0111

Der neue Wert von Register 5 lautet 1011 1011 1101 0111.

2. Beeinflussung des OV-Bits durch WS-Verschiebeinstruktionen

Das Überflußbit des Statusregisters wird nur durch SLA beeinflusst, da hier das Vorzeichen Beachtung finden muß, denn SLA ist eine arithmetische Verschiebung.

3. Statusregister-Auswirkungen und Verschiebe-Ergebnisse

R0-Inhalt	R5-Inhalt	L> A> EQ C OV	neuer Wert
SLA R1,0 (verschoben wird um 15 Bit = niedrigwertigste 4 in R0)			
0000000011111111	1100101100001010		0000000000000000
SRC R1,4 (verschoben wird um 4 Bit. R0 ist ohne Bedeutung)			
0000000000000000	1111110000010000		0000000000000000
SLA R1,0 (verschoben wird um 4 Bit = niedrigwertigste 4 in R0)			
0000000001110100	0000110000000001		0000000000000000
SRL R1,0 (verschoben wird um 16 Bit = Sonderregel 0 in R0)			
0000000010010000	1010000000001000		0000000000000000
SRA R1,5 (verschoben wird um 5 Bit. R0 ist ohne Bedeutung)			
0000000000110111	0011111110100101		0000000000000000
SRC R1,0 (verschoben wird um 16 Bit = Sonderregel '0' in R0)			
0000000000000000	0001000000010000		0000000000000000
SLA R1,0 (verschoben wird um 16 Bit = Sonderregel '0' in R0)			
0000000000000000	1100111110100010		0000000000000000

Bezüglich der Verschiebeart hier nochmals eine Liste:

SLA Arithmetische Linksverschiebung.
SRA Arithmetische Rechtsverschiebung.
SRC Zyklische Rechtsverschiebung.
SRL Logische Rechtsverschiebung.

4. Vorzeichen in logischen und arithmetischen Instruktionen

Bei einer logischen Verschiebung wird das Vorzeichen wie alle anderen Bits behandelt. Freiwerdende Bitstellen werden durch binäre Nullen ersetzt.
Bei einer arithmetischen Verschiebung hingegen wird das Vorzeichen besonders behandelt. Die Veränderung hat Auswirkungen auf das Statusregister, und freiwerdende Bitstellen werden durch das Vorzeichenbit ersetzt.

5. EQ-Beeinflussung durch Verschiebe-instruktionen

Das EQ-Bit kann durchaus gesetzt werden, da jede Verschiebung dazu führen kann, daß der neue Registerwert >0000 ist. Schließlich bezieht sich ja EQ nicht auf den Unterschied vom alten zum neuen Wert, sondern auf den Unterschied zwischen dem neuen Wert und Null!

Beispiel:

SRL R1,3
R1 = 0000 0000 0000 0101

Die gesetzten Bits werden aus dem Register geschoben:

0000 0000 0000 0010
0000 0000 0000 0001
0000 0000 0000 0000

Das EQ-Bit wird gesetzt, denn R5 ist jetzt gleich 0.

LÖSUNGEN ZU KAPITEL 1.3

1. LIMI-Unterbrechung

Keine Unterbrechung für die Spritesteuerung sollte während der Benutzung von VDP-Routinen erlaubt sein, da Interrupts den VDP-Eingang verändern können.

2. Sprite-Attributen-Liste

Die Tabelle benötigt 128 Bytes. Jeder Sprite wird darin mit 4 Bytes codiert. Das erste Byte bestimmt die Y-Koordinate, das zweite die X-Koordinate, das dritte verweist auf das dem Sprite zugeordnete Muster in der Spritemustertabelle. Im vierten Byte sind die führenden 4 Bits für die Ein/Ausblendung, die letzten vier Bits für die Farbe verantwortlich.

3. Sprite-Sichtbarkeit

Ein Sprite mit einem Y-Wert kleiner als >FF und größer als >BE ist auf dem Bildschirm nicht sichtbar, da diese Koordinaten außerhalb des sichtbaren Bereichs liegen.

4. Sprite-Definition

Ein Wert von >D0 als Y-Koordinate hat zur Folge, daß der Sprite und alle nachfolgenden als undefiniert behandelt werden und auf dem Bildschirm nicht sichtbar sind.

5. Sprite-Programm:

```
        IDT  'SPRITE'
*
        DEF  SPRITE
        REF  VSBW,VMBW,VWTR
*
LKW     DATA >0000,>001F,>1010,>1010
        DATA >1F70,>7170,>7073,>1E0C
        DATA >0000,>0080,>8080,>80FE
        DATA >FEFE,>FEFE,>FEFE,>3C18
*
SPRATR  DATA >60E6,>801F
        DATA >D000
*
SPRMOV  DATA >00FC,>0000
*
OWNWS   BSS >20
*
*
SPRITE  LWPI OWNWS          *Eigene Register laden
        LI   R0,>0384       *R0 mit Farbtabellen-Basisadresse laden
        LI   R1,>1100       *Farbe in R1 kopieren
        BLWP @VSBW          *Farbe in Tabelle eintragen
*
        LI   R0,>0701       *Hintergrundfarbe laden
        BLWP @VWTR          *Ans VDP-Register uebergeben
*
        LI   R0,>E201       *R0 laden
        MOVB R0,@>83D4      *uebertragen
        SWPB R0
        BLWP @VWTR
```

```
*
      LI   R0,>0400     *R0 mit Musterbeschreibungs-Basisadresse laden
      LI   R1,LKW       *Laden des Raumschiffmusters
      LI   R2,>20       *>20 Musterbytes zu schreiben
      BLWP @VMBW        *Musterbytes in Tabelle eintragen
*
      LI   R0,>0300     *R0 mit Sprite-Attributen-Basisadresse laden
      LI   R1,SPRATR    *R1 mit Attributen-Pointer laden
      LI   R2,>5        *>5 Bytes zu schreiben
      BLWP @VMBW        *Bytes in Tabelle eintragen
*
      LI   R0,>07B0     *R0 mit Spritebewegungs-Basisadresse laden
      LI   R1,SPRMOV    *R1 mit Bewegungs-Pointer laden
      LI   R2,>4        *>4 Bytes zu schreiben
      BLWP @VMBW        *Bytes in Tabelle eintragen
*
      LI   R1,>0100     *1 Sprite in Highbyte laden
      MOVB R1,@>837A    *und in Bewegungsmengenspeicher kopieren
*
NEXT  LIMI 2            *Unterbrechungssequenz
      LIMI 0
      JMP  NEXT
*
      END
```

ECONOMY - Das Wirtschaftsspiel

Haben Sie nicht auch schon einmal das Verlangen danach verspürt, ein Land zu regieren? Dann sollten Sie ECONOMY spielen!

Sie bauen Siedlungen, Schulen, Fabriken, Kraftwerke, Krankenhäuser, Festungen. Sie pflanzen Getreide an, bauen eine Viehzucht auf, lassen Kriegsschiffe und Fischerboote zu Wasser. Von Ihnen bezahlte Rebellen verhindern, daß Ihrem Gegenspieler das Gleiche gelingt.

Das Wetter ist unberechenbar: Sonne, Regen und Gewitterschauer verderben Ihnen die Ernte, setzen Siedlungen unter Wasser oder Blitze schlagen in Kraftwerke ein!

Für alles verteilt der Computer Plus- oder Minuspunkte - und am Ende eines Regierungsjahres wird Bilanz gezogen: Wer war der bessere Herrscher???

Jeweils mit ausführlicher Anleitung:

Cassette für TI-99/4a + XB + Joysticks oder
Diskette für TI-99/4a + XB + Speichererw. + Joysticks
nur DM 24.90

LöSUNGEN ZU KAPITEL 1.4.

1. Festlegung des Grafik-Modus

Der Grafik-Modus wird festgelegt, wenn Bit 6 im VDP-Register 0 und die Bits 3 und 4 im VDP-Register 1 zurückgesetzt, d.h. auf Null gesetzt sind.

2. Farb-Tabelle

Die Farb-Tabelle beginnt normalerweise an >0380 im VDP-Ram. Die Farbe der Buchstaben 'H' bis 'O' lässt sich verändern, indem die Farbcodierung des entsprechenden Character-Satzes geändert wird. Diese befindet sich in >0389. Das erste Nybble bestimmt die Vordergrundfarbe, das hintere die Hintergrundfarbe. Folgende Sequenz erlaubt es, die Farben auf schwarz/weiß zu setzen:

```
.
.
.
      LI R0,>0389
      LI R1,>1F00
      BLWP @VSBW
.
.
.
```

3. Bildschirm im Grafik-Modus

Der Grafik-Modus hat 24x32=768 Bildschirmpositionen. Lediglich im Basic werden bei Display und Print die ersten und letzten beiden Spalten nicht benutzt.

4. Möglichkeiten des Grafik-Modus

Die Benutzung von Sprites und automatischer Spritebewegung ist ohne weiteres möglich. Auch verschiedene Schriftfarben sind gestattet. Wollen Sie allerdings gleichzeitig verschiedenfarbige Schrift auf den Bildschirm bringen, ist ein entsprechendes Programm erforderlich.

5. VDP-Ram

Das VDP-Ram umfasst 16384 Bytes, was 4Kb entspricht.

6. Bildschirmfarbe

Die Bildschirmfarbe wird in VDP-Register 7 festgelegt, und zwar im niedrigwertigen Nybble.

7. Änderungen in VDP-Register 1

Um VDP-Register 1 zu ändern, ist eine Kopie des neuen gewünschten Wertes in Adresse >83D4 zu schreiben, weil die automatische Bildschirmabschaltung nach dem Drücken einer Taste den Inhalt dieser Adresse in VDP-Register 1 überträgt.

8. Bildschirmausgabe im Grafik-Modus

```
       DEF START
       REF VMBW
*
TEXT   'ASSEMBLER'
       EVEN
*
START  LI R0,>0180
       LI R1,TEXT
       LI R2,9
       BLWP @VMBW
*
LOOP   NOP
       JMP LOOP
```

9. Verwendung der Modi im Basic

Die unterschiedlichen Modi sind im Basic nicht ohne weiteres nutzbar, da es keine Möglichkeit gibt, von dort aus bestimmte VDP-Register bewusst zu ändern und so diese Modi zu initialisieren.

LÖSUNGEN ZU KAPITEL 1.5.

1. - 6. Text-Modus Aufgaben

Eine Lösung zu diesen Problemen würde über den Aufgabenbereich dieses Buches hinausgehen. Wir verweisen deshalb auf unsere im Anhang angebotenen Diskettenprogramme.

LÖSUNGEN ZU KAPITEL 1.6.

1. Wahr oder Falsch?

Aufgabe	wahr	Falsch	Begründung, soweit erforderlich
a		x	
b	x		
c	x		
d		x	
e		x	
f	x		
g	x		
h		x	
i	x		
j	x		
k		x	
l	x		Durch Änderung von VDP-Reg. 4!
m	x		
n		x	
o		x	
p	x		48x64=3072 3072/2=1536.
q	x		
r		x	JNC betrifft das Carry-Bit.
s		x	Die Musterbeschreibungstabelle steht nicht zur Verfügung.
t	x		

2. SZCB und SOCB

Wenn Sie sich hier nicht sicher sind, sollten Sie die ersten beiden Kapitel dieses Kurses wiederholen. Dort werden diese Befehle genau erläutert und es setehen auch Übungsaufgaben zur Verfügung.
SZCB sezt Bits im Zieloperanden, die sowohl im Quell- als auch im Zieloperanden eine 0 enthalten; allerdings nur im 1. Byte des 16-Bit-Wortes.
SOCB verfährt genauso mit den übereinstimmenden Einsen.

3. Sprites im Multicolor-Modus

Sprites im Multicolor-Modus sind sehr einfach zu realisieren. Die im Kapitel SPRITES IN MASCHINENSPRACHE genannten Grundsätze gelten auch hier.

Beachtet werden muß allerdings, daß bestimmte Tabellen, die bei der Verwendung von Sprites benötigt werden, an anderer Stelle beginnen (müssen). Dies gilt insbesondere für die Verwendung der Utility-Routinen VSBW etc.!

4. Multicolor-Möglichkeiten

Der Multicolor-Modus ist nicht für die Darstellung von Text und für feine Pixelgrafiken geeignet.

LÖSUNGEN ZU KAPITEL 1.7.

1. Grenzen von Hochauflösender Grafik

Die Möglichkeit, hochauflösender Grafik im Basic darzustellen, ist begrenzt, da hier nur die ASCII-Codes umdefiniert werden können, um eigene Zeichen zu erhalten, die dann wie hochauflösende Grafik wirken. Auch die Farbgebung ist beschränkt, da immer 8 aufeinanderfolgende ASCII-Character zu einer abhängigen Einheit zusammengefasst werden. Ein ähnliches Problem tritt damit auch im Grafik-Modus mit dem Assembler auf.

2. Initialisierung des Bit-map-modus

Der Bit-Map-modus wird durch Setzen von Bit 6 im VDP-Register 0 aktiviert.

3. Bildschirmdarstellungstabelle im Bit-Map-modus

Die Bildschirmdarstellungstabelle muß, wie im Grafik-Modus auch, auf 768 Zeichen Länge initialisiert werden. Alle Positionen werden mit einem bestimmten Character belegt. Begonnen wird in der oberen linken Ecke mit >00. Es wird von links nach rechts und von oben nach unten numeriert, bis >FF erreicht ist. Danach wird wieder mit >00 begonnen. Dieser Vorgang wird dreimal durchgeführt, bis der Bildschirm in drei gleiche Zonen, welche die Character >00 bis >FF enthalten, eingeteilt ist.

4. Startadresse der Tabelle

Wenn Sie ohne Sprites starten, sollte die Bildschirmdarstellungstabelle an >1800 des VDP-Ram beginnen.
Wollen Sie Sprites verwenden, dann empfiehlt sich ein Start dieser Tabelle bei >1C00, um die Möglichkeit zu haben, die Spritetabellen ab >1800 zu setzen.

5. Veränderung der Startadresse

Die Startadresse der Bildschirmdarstellungstabelle wird geändert, indem man einen neuen Wert in VDP-Register 2 schreibt. Der Inhalt dieses Registers multipliziert mit >400 ergibt die Startadresse der Bildschirmdarstellungstabelle.

6. Musterbeschreibungs- und Farbtabelle

Musterbeschreibungstabelle und Farbtabelle umfassen jeweils einen Bereich von >1800 Bytes im Bit-map-Modus.

7. Startadressen

Jede dieser Tabellen kann an >0000 oder >2000 beginnen. Wenn die Musterbeschreibungstabelle an >0000 beginnt, dann muß die Farbtabelle an >2000 beginnen - und umgekehrt.
Die Startadresse der Musterbeschreibungstabelle ergibt sich aus dem Inhalt von VDP-Register 4, multipliziert mit >800.
Die Startadresse der Farbtabelle wird errechnet aus dem Inhalt von VDP-Register 3, multipliziert mit >40.

8. Farbbestimmung im Bit-Map-Modus

Ein Wert der Farbtabelle gilt jeweils für die gesetzten oder für die nicht gesetzten Pixel in acht nebeneinanderliegenden Pixelspalten, analog zu der Definition in der Musterbeschreibungs-Tabelle.

9. Möglichkeit der Pixelgrafik

Bildschirmdarstellung, Musterbeschreibung und Farbtabelle werden jeweils in drei Teile geteilt. Welcher Teil angesprochen wird, ergibt sich aus der angesprochenen Adresse im VDP-Ram. Dabei wird mit 3x255 Charactern gearbeitet, die unabhängig voneinander mit Mustern und Farben codiert werden können. Dies erklärt auch die Notwendigkeit, die Bildschirmdarstellung vor dem Gebrauch des Bit-Map-Modus zu initialisieren.

10. Korrespondenz von Mustern und Farben.

Ausgehend von den unterschiedlichen Startadressen der Farbtabelle und der Musterbeschreibungstabelle, befinden sich die Mustercodierung und die Farbcodierung im selben Abstand zu der spezifischen Startadresse.
Das erste Nybble des Bytes in der Farbtabelle bestimmt die Vordergrundfarbe, also die Farbe der durch die Musterbeschreibungstabelle gesetzten Pixel. Das zweite Nybble des Bytes in der Farbtabelle bestimmt die Hintergrundfarbe, also die Farbe der nicht gesetzten Pixel.

11. Spritemustertabelle

Die Startadresse der Spritemuster-Tabelle wird durch den mit >800 multiplizierten Inhalt von VDP-Register 6 festgelegt.

12. Benutzung von VDP-Register 5

VDP-Register 5 legt die Startadresse der Sprite-Attributen-Liste fest. Um die Adresse zu erhalten, muß man den Inhalt des Registers mit >80 multiplizieren.

13. Veränderung eines VDP-Registers

Die Anweisung in Aufgabe 13 verändert die Startadresse der Sprite-Attributen-Liste. >05.. bezeichnet das Register; >..6C muß mit >80 multipliziert werden, um die neue Startadresse zu erhalten. Die Anweisung setzt also die Startadresse auf >3600 im VDP-Ram.

14. Möglichkeiten des Bit-Map-Modus

a) Die Darstellung von Sprites ist möglich.
b) Automatische Spritebewegung ist nicht möglich.
c) Die Ausgabe von Zahlen, Zeichen und Buchstaben ist nur durch ein Programm möglich, welches die entsprechenden Pixel einzeln setzt.
d) An >0000 kann nur die Musterbeschreibungstabelle oder die Farbtabelle beginnen.
e) Die Farbtabelle kann, muß aber nicht bei >0000 starten.
f) Natürlich; durch das niedrigwertige Nybble in VDP-Reg. 7!

15. Bildschirm löschen

Der Bildschirm muß durch Zurücksetzen aller Bits in der Musterbeschreibungstabelle und in der Farbtabelle erfolgen. Wenn Sie durch Löschung der Bildschirmdarstellungstabelle den Screen leeren, geht die Initialisierung verloren. Bei einem anschließenden neuen Aufbau des Bildschirms würde auch trotzdem wieder die alte Muster- und Farbdefinition sichtbar.

16. Löschen der Bildschirmdarstellung

Die Bildschirmdarstellung kann nach wie vor nur durch Löschen der Bildschirmdarstellungstabelle geschehen. Alle anderen Punkte löschen nur Muster, Farben oder ähnliches. Dies ist jedoch unabhängig davon, daß ein wirklich 'cleaner' Bildschirm nur durch Löschen von Musterbeschreibungs- und Farbtabelle erreicht wird.

17. Sinnvolles Löschen

Ein wirklich leerer Bildschirm wird durch Löschen der Farbtabelle und der Musterbeschreibungstabelle erreicht.

18. Programm zum löschen

```
        DEF START
        REF VSBW
*
START   LI R0,>0000
        LI R1,>0000
L1      BWLP @VSBW
        INC R0
        CI R0,>1800
        JNE L1
*
        LI R0,>2000
L2      BLWP @VSBW
        INC R0
        CI R0,>3800
        JNE L2
  .
  .
  .
```

19. Setzen von Punkten

Um einen Punkt im Bit-Map-Modus zu ändern, muß ein Byte in der Musterbeschreibungstabelle geändert werden.

20. Farbveränderung von Punkten

Um einen bestimmten Punkt zu färben, muß das führende Nybble in dem Byte der Farbtabelle geändert werden, welches analog in der Musterbeschreibungstabelle die Mustercodierung enthält.

21. Routine zum Färben:

Da die Farben mit der Musterdarstellung, bezogen auf die jeweils unterschiedliche Basisadresse beider Tabellen, korrespondieren - das heißt, die Pixelzeilenmustercodierung eine ganz bestimmte Byteposition einnimmt, welche in der Farbtabelle genau die Farbe der Pixelzeile (8 nebeneinanderliegende Pixel) codiert, ist es sehr einfach, anhand der Bildschirmposition die Farb-Bytes abzuleiten. Anstelle von 'linker' und 'rechter' Byteblock müssen hier natürlich Vorder- und Hintergrundfarbe berücksichtigt werden.

Wir ersparen uns an dieser Stelle ein entsprechendes Programm, da im Text des Kurses ausführlich auf die Darstellungsformen von Mustern und Farben im Bit-Map-Modus eingegangen wird.

22. Musterbeschreibung

Die Koordinaten, die durch Byte 300 in der Musterbeschreibungstabelle geändert werden sind folgende:

X 32-39
Y 11

Zum Aufbau der Musterbeschreibungstabelle finden Sie eine Tabelle mit den Bytes in Kapitel 1.7., die Sie nur entsprechend zu erweitern brauchen, um die Koordinaten zu erhalten.

23. Programmauswirkung

Das Programm setzt den Bit-Map-Modus und initialisiert die Bildschirmdarstellungstabelle, um in einer Endlosschleife zu enden.

24. Koordinaten-Beeinflussung mit Bytes.

Für den Start an Adresse >2000 ist dieser Wert zu addieren.

X	10	103	11	99	20	204	28
Y	54	22	22	80	56	190	115

25. Farb-Beeinflussung

Die Farb-Tabelle beginnt an >2000. Für den Start an >0000 ist >2000 (8192) zu subtrahieren. Der Wert in Register 6, also die Farbe als solche, ist ohne Belang.

R4=	10	103	11	99	20	204	28
R5=	22	80	54	115	190	80	22

26. Wahr oder Falsch

Aufgabe	wahr	falsch	Begründung, soweit erforderlich
a		X	24x40 Punkte = Text-Modus
b	X		Sie sind aber immer nur als 8x8 Block ansprechbar, also 24x32 Positionen.
c	X		
d	X		
e	X		
f	X		Nacheinander, nicht zugleich!!!
g		X	
h		X	
i	X		
j		X	
k	X		
l		X	Im Programm werden nur die ersten 768 Positionen gelöscht. Der Text-Modus hat aber 960!
m	X		
n		X	Die Definition ist 8x8 Pixel, nur bei der Darstellung auf dem Bildschirm werden die letzten beiden Pixeleingänge ignoriert.
o		X	Denken Sie an Extended Basic!
p		X	VDP-Register 0 muß geändert werden, aber schreiben Sie keine Kopie in >83D4!!!
q		X	Die Farbe wird durch VDP-Register 7 bestimmt.
r	X		

27. Benutzung der VDP-Register

Die Startadressen errechnen sich wie folgt:

Bildschirmdarstellungstabelle:
Inhalt von VDP-Register 2 multipliziert mit 1024.

Farbtabelle:
Inhalt von VDP-Register 3 multipliziert mit 64.

Sprite-Attributen-Liste:
Inhalt von VDP-Register 5 multipliziert mit 2048.

Musterbeschreibungstabelle:
Inhalt von VDP-Register 4 multipliziert mit 128.

Sprite-Beschreibungstabelle:
Inhalt von VDP-Register 6 multipliziert mit 2048.

28. Modus-Umschaltung

Ein komplettes Programm zur Modus-Umschaltung, allerdings ohne die Möglichkeit, dies direkt in Maschinensprache per Tastendruck zu erreichen, finden Sie in Kapitel 3 durch das dort auszugsweise veröffentlichte Programm MODE CONTROL.

29. Eignungen der Modi

a) Dateiprogramm: Textmodus (evtl. Grafik-Modus)
b) Balkendiagramme: Multicolor-Modus
c) Schnittmusterzeichnungen: Bit-Map-Modus
d) Grundrißzeichnungen: Bit-Map- oder Multicolor-Modus
e) Spiele mit farbe und Bewegung: Grafik- oder Bit-Map-Modus
f) Musikprogramm: beliebig, da unabhängig!
g) Grafik-Tablett: Bit-Map-Modus (evtl. Multicolor-Modus)
h) Titelbilder mit großen Buchstaben: Multicolor-modus

zu h) Die Zeichen müssen aus den einzelnen Punkten zusammengesetzt werden. Dies gibt im Multicolor-Modus sehr schöne Effekte! Probieren Sie es doch einfach einmal aus!

LÖSUNGEN ZU KAPITEL 1.8.

1. Aufruf von Maschinenprogrammen aus dem Basic

Der Aufruf erfolgt mit CALL LINK("Programmname")

2. Starten von Maschinenprogrammen aus dem Basic

Im Maschinenprogramm müssen eigene Register verwendet werden, da der Basic-Interpreter von sich aus ebenfalls Register verwendet und diese bei der Rückkehr vom Basic-Programm nicht verändert sein dürfen. Mit LWPI wird der entsprechende Registersatz angezeigt. Bei der Rückkehr ins Basic ist außerdem das GPL-Status-Byte zu löschen und eine der gezeigten Möglichkeiten als Rücksprung zu programmieren. Einsprungstellen sind in einer DEF-Anweisung anzugeben.

Im Basic muß mit INIT die Benutzung von Maschinenprogrammen vorbereitet werden, das MC-Programm mit LOAD geladen und zum Schluß mit LINK gestartet werden. INIT und LOAD sind nur einmal erforderlich, auch wenn das MC-Programm mehrmals oder über verschiedene mit DEF definierte Eingangsstellen angesprochen werden soll.

3. Deutscher Zeichensatz

Eine Lösung zu dieser Aufgabe, in etwas abgeänderter Form, finden Sie in Kapitel 3 TORPEDO BASIC. Die dort veröffentlich- ten Programmteile stellen auch den deutschen Zeichensatz zur Verfügung.

4. Register-Startadresse

Die Konsolen-Programme, wie auch der Basic-Interpreter, benutzen den Bereich ab >83E0 als Arbeitsregister.

5. USRWS

Das Symbol USRWS kennzeichnet den Arbeitsregisterbereich, welcher bei einem Aufruf aus dem TI-Basic als Benutzer- Registerbereich geladen wird. Dieser Bereich kann dann für eigene Zwecke als Registerbereich behandelt werden. Es ist jedoch nicht zwingend, diesen Bereich zu benutzen.

6. Rücksprünge zum Basic

a) Durch RT
b) Durch B
c) Durch Verzeigung zu >0070 durch B §>0070

7. Schreiben im Text-Modus aus dem Basic

Die Lösung zu dieser Aufgabe finden Sie im Kapitel 3. Mode Control ermöglicht das Ansprechen des Text-Modes und damit die Ausgabe von Text auf den Bildschirm im Text-Modus! Dies ist allerdings komfortabler, als es die Aufgabenstellung erwartet.

8. Veränderungen von Register 11

Eine Veränderung dieses Registers kann durch Sicherung des Registerbereichs oder durch Benutzung von Anwenderregistern verhindert werden.

9. EQU

Die Assembler-Direktive EQU setzt ein Symbol gleich zu einer Adresse, sodaß anstelle der Adresse künftig auch das Symbol verwendet werden kann.

Beispiel: ADR EQU >A240

Anstelle von >A240 kann ab sofort auch ADR geschrieben werden, statt §>A240 auch §ADR, und auch in Ausdrücken kann selbstverständlich das Symbol benutzt werden.

10. BSS

Die Assembler-Direktive BSS reserviert einen bestimmten Speicherblock, der mit dem vor BSS genannten Symbol bezeichnet ist. Die Zahl hinter BSS gibt die Byte-Zahl an, die reserviert wird.

Beispiel: BLOCK BSS >20

Der Befehl reserviert einen 32 Byte großen Bereich, beginnend an Speicherstelle BLOCK. Angenommen, BLOCK befindet sich bei >A200, dann werden die Bytes der Adressen >A200 bis >A21F reserviert.

11. LWPI

Die Assembler-Mnemonic lädt den 'Arbeitsspeicher-Zeiger'. Dadurch ist es möglich, verschiedene Bereiche festzulegen, die dann nach entsprechendem LWPI als Registerbereich behandelt werden. Die Adresse hinter LWPI gibt die Startadresse des Registerbereichs an.

Beispiel: WS LWPI MYREG

Der Befehl definiert den Registerbereich an der Adresse, an der sich das Symbol MYREG befindet. MYREG muß natürlich vorher am besten mit EQU oder BSS definiert werden.

12. Wahr oder Falsch

Aufgabe	wahr	falsch	Begründung, soweit erforderlich
a		X	Der Befehl setzt USRWS nur gleich zu der Adresse >20BA, lädt aber nicht den Registerbereich. Dafür ist LWPI zuständig
b	X		
c		X	Es wird ein File mit dem Namen START geladen, der zwar auch ein MC-programm mit dem Namen START enthalten kann, dies aber nicht unbedingt muß.
d	X		
e	X		
f		X	Es müsste heißen: B §>0070
g	X		
h		X	Es müsste heißen: B §DORT

13. ASCII-Offset

Der ASCII-Offset ist ein fester Wert, welcher bei Bildschirmausgaben über Maschinenspracheroutinen, die aus dem Basic heraus aufgerufen werden, addiert werden muß, um die gewünschten Zeichen zu erhalten, da verschiedene Tabellen des VDP-Ram in den Basic-Versionen die gleichen Startadressen haben und bestimmte Bytes relativ zu dieser Startadresse angesprochen werden müssen.

14. Zeichendarstellung über Basic-Aufruf

Zeichen können durch eine VSBW-Schleife dargestellt werden, welche x-mal (x=Stringlänge) einen Character des Strings um den Offset (>60) erhöht und dann auf dem Bildschirm ausgibt. VMBW kann nicht verwendet werden.

LÖSUNGEN ZU KAPITEL 1.9.

1. Wahr oder Falsch?

Aufgabe	wahr	falsch	Begründung, soweit erforderlich
a	X		
b		X	
c	X		
d		X	
e		X	Wenn es sich nicht um eine Ordnung handelt, muß der Inhalt von R0 Null sein!
f		X	Der Wert von R0 muß nicht Null sein, wenn es sich um eine Ordnung handelt.
g	X		
h		X	Der erste ASCII-Code steht an ADR+1!
i	X		
j	X		Maximal 16 Parameter können in einem LINK übergeben/übernommen werden.
k	X		
l		X	Die Maximale Stringlänge ist 255 Bytes; ein Byte wird zur Angabe der Stringlänge benötigt und steht nicht zur Verfügung.

2. Die BSCSUP-Utility-Routinen

In den BSCSUP-Utilities der E/A-Diskette Part A sind folgende Routinen

enthalten: STRREF, NUMASG, ERR, STRASG und NUMREF.

nicht enthalten: VWTR, VMBW, XMLLNK und VSBW.

3. Programmauswirkung

Hereingefallen? Durch den LINK-Befehl wird natürlich nur START und nicht WORT durchlaufen. WORT im Link-Befehl ist lediglich ein Parameter. Daß 'zufällig' auch eine definierte Eintrittsstelle im Maschinenprogramm so heißt, ist ohne Belang. Für das Ausführungsresultat ist also lediglich alles das verantwortlich, was hinter START steht.
Der Programmteil START liest den Stringparameter "WORT" ins Maschinenprogramm und bringt diesen in die obere linke Ecke des Bildschirms.

4. Übergabe von Ordnungen

Im Maschinenprogramm muß im Register 0 die Elementenzahl angegeben sein. Abgesehen davon sind natürlich alle anderen Vorbereitungen für die Übernahme von Parametern zu treffen.

Im Basic muß im LINK-Befehl die Ordnung mit ihrem Namen, den Klammern und einer Anzahl Kommata zwischen den Klammern zur Kennzeichnung der Ausdehnung angegeben werden.

Beispiel: CALL LINK("START",X$(,,))

Der befehl startet das Maschinenprogramm START, wobei die dreidimensionale Ordnung X$ übergeben wird.

5. Berechnung der Elementenzahl

Aus einem dreidimensionalen Array, hier F$(10,5,3), ergeben sich die Elementenzahlen für die einelnen Elemente, wenn die Option-Base im Basic auf 0 gesetzt ist (Standartgröße), wie im Kustext beschrieben. Lesen Sie dort nochmals nach. wenn Sie nicht weiterkommen.

6. Berechnung der Elementenzahl

Aus einem dreidimensionalen Array, hier F$(10,5,3), ergeben sich die Elementenzahlen für die einelnen Elemente, wenn die Option-Base im Basic auf 1 gesetzt ist, aus der im Text genannten Berechnungsformel. Wir ersparen uns daher eine ausführliche Lösung.

7. Übergabe von Ausdrücken

Um einen Ausdruck vom Basic an ein Maschinenprogramm zu übergeben, wird dieser an die entsprechende Parameterstelle des LINK-Befehls geschrieben. Der Ausdruck wird vom Basic-Interpreter ausgerechnet und dann erst an das Maschinenprogramm übergeben.

8. Basic-Anweisungen

CALL LINK("PROG",C$,"DOMINO") ruft ein Maschinenprogramm mit dem Namen PROG auf, wobei die Parameter C$ (variable) und "DOMINO" (konstant) es durchlaufen.

9. - 10. Programm 'Verkehrtherum' und Programm 'Senkrecht'

Sie haben warscheinlich den richtigen Lösungsweg zur Darstellung gefunden. Natürlich kam es uns hierbei nicht so sehr auf die Parameterübergabe als auf die richtige Offset-Addierung an, ohne die Sie nicht das gewünschte Ergebnis auf dem Bildschirm erhalten.

Sollten Sie den Offset vergessen haben, schauen Sie sich noch einmal Kapitel 1.8. an!

11. Wahr oder Falsch?

Aufgabe	wahr	falsch	Begründung, soweit erforderlich
a		x	Register 1 enthält die Parameterposition
b	x		
c		x	siehe Begründung a)
d		x	NUMREF ist in BSCSUP enthalten
e		x	
f		x	Wie kommen Sie denn darauf, daß dies richtig sein könnte???
g	x		In einem 2-Byte-Wort kann maximal eine Zahl dieser Größe dargestellt werden.
h		x	>834A ist richtig, nicht aber 'ASCII'!
i		x	XMLLNK ist eine ROM-Routine
j	x		Startadresse von CFI.
k		x	Die Startadresse an XMLLNK muß in unmittelbar folgenden Data-Zeilen übergeben werden.
l		x	Die Symbole müssen von Ihnen definiert werden. Man benutzt sie nur zur leichteren Verständlichkeit.
m	x		
n	x		
o		x	Es wird immer der gleiche Bereich als Parameter-Buffer benutzt, daher müssen übernommene Parameter gesichert werden, indem man sie an amdere Adressen überträgt.
p	x		

12. Multiplikation mit LINK

Das Programm stellt kein Problem dar, solange die Parameter ganzzahlig und nicht kleiner als -32768 und nicht größer als +32767 sind. Auch das Ergebnis sollte sich in diesem Rahmen bewegen. Für die Multiplikation stellt XMLLNK übrigens eine eigene Routine ähnlich CIF und CFI zur Verfügung. Schlagen Sie hierzu bitte im Handbuch unter XMLLNK nach.

13. Zahlenablage in FAC

Die Zahlendarstellung erfolgt in der RADIX 100 SCHREIBWEISE. Im ersten Byte von FAC befindet sich der Exponent, in den folgenden wird die Zahl dargestellt.

14. Umwandlung mit XMLLNK

Nach der Umwandlung befindet sich die Zahl in Hexadezimal-schreibweise in >834A und >834B. Den Inhalt dieser beiden Bytes können Sie nun beliebig weiter bearbeiten.

15. Stringdisplay an variablen Koordinaten

Das größte Problem wird hier wohl die Umwandlung von Zeile und Spalte in eine absolute Bildschirmadresse sein. Dies ist jedoch mit dem SLA-Befehl leicht zu bewerkstelligen, wenn Sie 32 Zeichen pro Zeile verwenden. Ansonsten muß MPY verwendet werden. Die Ausgabe des Strings selbst dürfte keinerlei Probleme aufwerfen. Wenn doch, sollten Sie die Kapitel 1.8. und 1.9. wiederholen.

16. Quadrat

Zu dieser Aufgabe bieten wir noch keine fertige Lösung an. Wir wollen Ihnen aber einen Lösungsweg zeigen.
Wir setzen voraus, daß Sie die nötigen Vorbereitungen getroffen haben. Sie lesen den Parameter ein (NUMREF), wandeln ihn in einen Integerwert um (CFI) und multiplizieren ihn mit MPY mit sich selbst, also MPY R4,R4, wenn sich der Multiplikator in Register 4 befindet. Danach muß das Ergebnis, welches jetzt (solange es nicht Größer als 32767 ist) in R5 enthalten ist, wieder an FAC übergeben, in eine Fließkommazahl zurückverwandelt (CIF) und ans Basic übertragen werden (NUMASG).

17. BEG$-Ersatz

Auch zu dieser Aufgabe an dieser Stelle nur einen Lösungshinweis, denn Sie sollten inzwischen die einzelnen Schritte schon eigenständig programmieren können.
Zuerst wird der Stringparameter geholt und in einem Buffer, BUFF genannt, gespeichert. Dann übernimmt man die Ganzzahlwerte WERT1 und WERT2, wobei WERT1 die Startstelle und WERT2 die Endstelle enthält. Die Differenz zwischen WERT1 und WERT2 ist unsere Stringlänge, die wir ins Längenbyte von BUFF schreiben müssen. Anschließend müssen die Bytes des Teilstrings, der an BUFF+WERT1 beginnt und bis BUFF+WERT2 reicht, in den Bereich BUFF+1 bis zur Stelle BUFF+(WERT2-WERT1) verschoben werden, am besten mit MOV in einer kleinen Schleife. Zu guter letzt muß der neue String an die Variable im LINK-Befehl übergeben werden.
So kompliziert dies auf den ersten Blick scheinen mag - eigentlich ist es doch ganz einfach, oder??

18. Löschen von Bildschirmausschnitten

Die Lösung zu diesem Problem bieten wir, allerdings nicht einsehbar für den Benutzer, auf unserer Programmdiskette TORPEDO BASIC (vorgestellt in diesem Buch). Das Problem ist auch hier wieder die Berechnung der absoluten Startadresse des Löschens auf dem Bildschirm. Außerdem muß der 'Übersprung', also die Strecke zwischen rechter Fensterbegrenzung und linker Begrenzung eine Zeile tiefer, berechnet werden. Das Löschen auf der Y-Achse kann über eine Schleife erfolgen.

19. Umschalten auf den Text-Modus im Basic

Eine Lösung zu diesem Problem finden Sie im Ausschnitt des Programms MODE CONTROL im Softwareteil, Kapitel 3.

20. Kleinstes gemeinsames Vielfaches.

Lösungshinweis: Die beiden Parameter werden zunächst auf Gleichheit verglichen. Sind sie ungleich, wird der jeweils kleinere Wert um den Ursprungswert (bei ersten Mal also um sich selbst), vermehrt. Dann wird erneut verglichen und ggf. der kleinere vermehrt. Sind beide verglichenen Werte gleichgroß, ist das 'KGV' gefunden. Dieser Wert muß dann ans Basic zurückgegeben werden.

21. Zahlen-Umwandlung

Das gebräuchliche Symbol für die Routine, welche eine Integerzahl in eine Fließkommazahl umwandelt, lautet CIF.

22. Übergabe von Ordnungen an das Basic

Um Ordnungen vom Machinenprogramm an das Basic zu übergeben, müssen folgende Schritte getan werden:

- Register 0 muß die Elementenzahl enthalten.
- Register 1 muß bei numerischen Parametern die Parameterposition im LINK-Befehl; bei Stringparametern die Adresse, an der sich der zu übergebende String befindet, enthalten.
- Register 2 muß (nur bei Stringparametern) die Parameterposition im LINK-Befehl enthalten.
- Bei numerischen Parametern muß sich der zu übergebende Wert an Adresse >834A und folgende in der RADIX-100-Schreibweise befinden, was durch XMLLNK-CIF erreicht wird.
- Die Routinen NUMASG (numerisch) bzw. STRASG (String) übergeben die Parameter.

Fortsetzung zu Aufgabe 22.

- CALL LINK im Basic muß an der Stelle, wo die Ordnung übergeben wird, den Namen der Ordnung und in Klammern eingeschlossen eine Anzahl Kommata enthalten, welche die Dimensionierung ausdrückt, also zum Beispiel C$(,) für eine zweidimensionale Stringordnung.

23. Fehlerbereinigung

Die zwei Möglichkeiten, zu verhindern, daß falsche Werte in ein Maschinenprogramm gelangen, sind im Basic die Überprüfung auf Richtigkeit durch geeignete Routinen und im Maschinenprogramm die Verkettung mit ERR zur Fehlermeldungsroutine.

24. Fehlercode-Übergabe an ERR

Der Fehlercode, dessen Nachricht Sie auszugeben wünschen, ist an ERR im hochwertigen Byte von Register 0 zu übergeben. Eine Liste mit den Fehlercodes finden Sie in Kapitel 5.

25. DISPLAY-Ersatz

Eine Lösung zu diesem Problem ist im Ausschnitt des Programms MODE CONTROL enthalten, welchen Sie im Softwareteil (Kapitel 5) finden.

26. Utility-Rätsel

Numerische Parameter werden vom basic an ein Maschinenprogramm durch NUMREF übergeben. Wenn Sie diese Aufgabe nicht richtig gelöst haben, sollten Sie unbedingt das letzte Kapitel wiederholen, bevor Sie weiterarbeiten.

27. Programmauswirkung

Das Programm löscht, beginnend in der oberen linken Ecke des Bildschirms, soviele Positionen, wie Sie vom Basic her übergeben. Dabei wird von links nach rechts und von oben nach unten gelöscht.

28. Begriffserläuterungen

'Integer': Der Begriff Integer kennzeichnet eine ganze Zahl, das heißt, eine Zahl ohne Stellen nach dem Komma. 115 ist eine Integerzahl, 115.2 nicht.

'Fließkomma': Eine Fließkommazahl kann auch Stellen nach dem Komma besitzen. 115.2, 65.01 sind Fließkommazahlen. Fließkomma deshalb, weil das Komma nicht an eine bestimmte Position gebunden ist.

'Exponent': Der 'Exponent' ist der Wert, der kennzeichnet, mit dem wievielten Vielfachen einer Zahl gearbeitet werden soll. Ein Exponent von 3 bei einer Basiszahl von 10 besagt, daß mit 10 hoch 3, also 10x10x10 gearbeitet werden muß.

Fortsetzung zu Aufgabe 28.

'Radix-100-Schreibweise': Dies ist eine besondere Art der Zahlendarstellung. Dabei wird ein bestimmter Wert durch eine Fließkommazahl und einen Exponenten ausgedrückt, also Beispielsweise statt 156.2 schreibt man 15.62 x 10 hoch 1.

'Stapelspeicher': Ein Speicherbereich, der auf eine ganz bestimmte Art und Weise genutzt wird. Einen solchen Speicher muß man sich vorstellen wie Papierblätter in einem Ablagekorb. Die jeweils zuletzt hineingelegten Blätter/Bytes können zuerst wieder herausgenommen werden, da sie obenauf liegen.

Fortsetzung zu Aufgabe 28.

'Wertzuweisung': Unter einer Wertzuweisung versteht man die Zuordnung einer Konstanten zu einer Variablen, also im Basic zum Beispiel A=15, wobei der Wert 15 der Variablen A zugewiesen wird. In Assembler erfolgt eine Wetzuweisung entweder durch die Direktiven BYTE, DATA und TEXT oder durch Kombination von LI und MOV/MOVB etc. - je nachdem, wem was zugewiesen werden soll.

'Parameter': Ein Wert, der bei einer Verzweigung, Verkettung oder bei einem Sprung an einen anderen Programmteil übergeben wird. Im Basic erfolgt die Parameterübergabe zum Beispiel an ein Unterprogramm 'CALL CHAR(...,...)', in Assembler werden die Register dazu herangezogen, zum Beispiel R0 und R1 für VSBW.

'Ordnung': Eine Ordnung ist eine Variablengruppe, deren einzelne Elemente durch Angabe eines Parameters unterschieden werden. Eine solche Ordnung ist zum Beispiel F(,); diese enthält zum Beispiel die Elemente F(0,1), F(0,2), F(0,3), F(1,0), F(1,1), F(1,2) und F(1,3), wenn die Dimensionierung auf F(2,3) erfolgte. Ordnungen befinden sich 'geordnet' in einem bestimmten Speicherbereich.

'2er-Komplement': Die Darstellung von negativen Zahlen (Werte kleiner als 0) ist mit Binärzahlen/Hexadezimalzahlen normalerweise nicht möglich. Man hat sich daher geeinigt, daß das erste Bit in einem Wort bzw. Byte die Zahl als negativ kennzeichnet, wenn es gesetzt ist, und als positiv, wenn es (auf 0) zurückgesetzt ist. Außerdem sind alle üblicherweise gesetzten Bits auf 0 zurückgesetzt und umgekehrt. Um den tatsächlichen Wert zu ermitteln, muß das 2er-Komplement gebildet werden. Dazu werden alle gesetzten bits zurückgesetzt und alle zurückgesetzten gesetzt. Anschließend wird zu der erhaltenen Zahl 1 addiert. Beispiele hierzu finden Sie im Kapitel 1.1.!

29. Arbeiten mit Ordnungen

Wenn Ihr Programm nicht läuft, sollten Sie sich das Kapitel über die Übergabe von Ordnungen als Parameter nochmals ansehen. Die Lösung selbst ist für eine Darstellung an dieser Stelle zu komplex.

LÖSUNGEN ZU KAPITEL 1.10.

1. Schwierigkeiten im Extended Basic

Die Maschinenprogramme werden vom Lader des Extended Basic anders aufgenommen als vom Lader des Editor/Assemblers. Im Handbuch zum Editor/Assembler finden Sie dazu nähere Hinweise.

2. Utilities im Extended Basic

Die ROM-Utilities haben im Extended basic Modul teilweise andere Startadressen als im Editor/Assembler Modul. Außerdem sind die Symbole nicht vordefiniert. Deshalb müssen Maschinenprogramme, die im Extended Basic laufen sollen, am Anfang Gleichstellungen mit EQU zu den Utilityroutinen enthalten.

3. Eintrittsstellen

FAC, USRWS und GPLWS sind im Editor/Assembler an der gleichen Stelle wie im Extended Basic.

4. Systemabsturz

Der Maschinenprogramm-Lader des Extended Basic kann eventuell ein Basic-Programm überschreiben, wenn eine MC-Routine geladen wird, sodaß das Basic-Programm nicht mehr richtig ausgeführt werden kann.

5. Bestimmung der Startadresse

Sie können den Start eines Maschinenprogramms bestimmen, indem Sie die Adresse in einer AORG-Anweisung angeben. AORG steht für Absolute ORiGin (= soviel wie absoluter Ursprung). Hier ein Beispiel:

```
AORG >A000
```

Die Startadresse des MC-Programms wird auf >A000 festgelegt.

6. Speicherstelle der Utilityroutinen

Die Utilities befinden sich im Editor/Assembler und Extended Basic ab Adresse >2000.

7. MC-Programmbenutzung in Extended Basic

Zu einigen Übungsaufgaben finden Sie Lösungen im Softwareteil. Dort wird auch gezeigt, wie der Zugriff vom Extended Basic her möglich ist. Von allen Aufgaben hier eine Extended Basic Zugriffsmöglichkeit zu zeigen, würde den Rahmen dieses Buches beiweitem sprengen - was Sie aber nicht davon abhalten sollte, alle Aufgaben zu lösen!

8. VPEEK und VPOKE

Das Programm VPOKE ist als VDPOKE im Ausschnitt zu TORPEDO BASIC im Softwareteil (Kapitel 5) enthalten. VPEEK funktioniert im Prinzip entsprechend, nur das die Werte eben vom Maschinenprogramm ans Basic übergeben werden müssen.

LÖSUNGEN ZU KAPITEL 1.11.

1. Wege zum Speicherauszug

Um einen Speicherauszug zu erhalten, muß beim Assemblieren, wenn der Assembler geladen ist, hinter LIST FILE NAME eine entsprechende Datei angegeben werden, zum Beispiel DSK1.SPAUSZ! Ausserdem sind bei OPTION das S für Speicherauszug und L für Liste anzugeben. Der Speicherauszug kann auch direkt auf einen Drucker erfolgen, zum Beispiel mit PIO.

2. Speicherauszug auf den Drucker

Wenn der Speicherauszug auf Diskette bereits vorliegt, kann dieser mit der PRINT-Option des Editors ausgedruckt werden, wie Sie es vom Ausdruck der anderen Assembler-Dateien her gewohnt sind. Ansonsten ist der Ausdruck wie unter 1) gezeigt möglich.

3. Assemblieren per Hand

Haben Sie richtig assembliert? Nachfolgend der Speicherauszug zum Übungsprogramm.

4. Disassemblieren, I

Die Lösung dazu finden Sie im Speicherauszug von PARTISAN VILLAGE, der in diesem Buch angegeben ist.

5. Disassemblieren, II

Schwierigkeiten? - So sehr Sie sich auch bemühen, etwas ordentliches aus dem Speicherauszug herauszubekommen - es will nicht gelingen. Das liegt daran, daß wir nicht mit einer Speicherstelle begonnen haben, an der sich ein Mnemonic befindet, sondern mit irgendeiner anderen. Es ist also sehr wichtig zu wissen, wo nun eigentlich das Maschinenprogramm beginnt. Ist keine Eintrittsstelle angegeben, hilft nur probieren. Wenn die ersten beiden Bytes weggelassen werden, erhalten Sie ein brauchbares Ergebnis. Auch dieser Speicherauszug ist teil des Abdrucks von PARTISAN VILLAGE in diesem Buch!

6. Befehlsformate

Zur Lösung dieser Aufgabe lesen Sie bitte die ausführlichen Erläuterungen im Kapitel 1.11., welches die Formate betrifft.

7. Unterschiede der Formate

dto.

8. Auswirkungen von Direktiven auf den Speicherauszug

REF: Nimmt ein vordefiniertes Utility-Symbol mit in die Symbolliste auf.

DEF: Nimmt ein Symbol mit in die Symbolliste auf.

BYTE: Schreibt nachfolgende Byte-Daten in die Liste.

TEXT: Schreibt die jeweilige Textlänge in die Liste.

UNL: Unterbricht die Auflistung bis zur Direktive LIST.

9. Dollar-Zeichen

Das Dollarzeichen '$' symbolisiert in Programmen die augenblickliche Adresse, die der interne Programmzähler anzeigt. Dadurch werden relative Sprünge ohne Eintrittsstellenangabe möglich.

10. Poken von MC-Programmen

Es darf nicht vergessen werden, die Symboltabelle ebenfalls in die Speichererweiterung zu poken, und zwar an eine ganz bestimmte Stelle. Beachten Sie hierzu die entsprechenden Erläuterungen im Handbuch zum Editor/Assembler ab Seite 275.

LÖSUNGEN ZU KAPITEL 1.12.

1. CRU

'CRU' ist die COMMUNICATION REGISTER UNIT. Mit ihr haben Sie Zugriff auf die Konsolen-Roms und Peripherien. Die Bedeutung der CRU-Befehle können Sie ausführlich im Kapitel 1.12. und im Handbuch zum Editor/Assembler ab Seite 148 nachlesen.

2. CRU-Basis-Adresse

Die CRU-Basis-Adresse befindet sich im Register 12, Bits 3-14. Durch entsprechende Verschiebeinstruktionen können Sie diese ermitteln bzw. festlegen.

PARTISAN VILLAGE

Die Diskette mit dem kompletten Spiel aus ASSEMBLER KURS II!

Tolle Action, Sound, Grafik in reiner Maschinensprache.

Befreien Sie die Kleinstadt von den Partisanen...

Lassen Sie sich zum General befördern...

PARTISAN VILLAGE...

für TI-99/4a, Speichererweiterung, Diskettenlaufwerk und Joysticks mit

- Extended Basic Modul

Mit deutscher Anleitung nur DM 39.90

oder

- Editor Assembler Modul

Mit deutscher Anleitung nur DM 34.90

3

3. SOFTWARE

In diesem Kapitel stellen wir Ihnen zwei Routinen vor, welche Basic bzw. Extended Basic unterstützen bzw. erweitern. Zum Einen handelt es sich dabei um die Möglichkeit zur Umschaltung des Modus, zum Anderen um Ersatz für die Befehle PEEKV und POKEV.

Über unseren Vertrieb sind verschiedene Systemerweiterungen erhältlich, welche diese Möglichkeit der MC-Unterstützung nutzen. Fordern Sie weitere Informationen an über:

- TORPEDO BASIC

- MODE CONTROL

- TEXT UTILITIES

MODUB-UMSCHALTUNG

Die folgende Routine ist aus dem Basic mit dem EDITOR/ASSEMBLER Modul und den zuvor geladenen BSCSUP-Utilities ansprechbar. Sie ermöglicht das Umschalten in den TEXT-Modus mit

CALL LINK("MODE",2)

und das Zurückschalten in den Grafik-Modus mit

CALL LINK("MODE",1).

Eine Fehlerüberprüfung erfolgt nicht. Beachten Sie, daß die im Basic eingebauten Routinen PRINT, DISPLAY, INPUT, ACCEPT etc. nicht für den Text-Modus ausgelegt sind und dort nicht richtig funktionieren. Stattdessen sind besondere Routinen zu entwerfen (unserer Diskette MODE CONTROL stellt alle notwendigen Befehle zur Verfügung).

```
        IDT  'MODECON'
*
        DEF  MODE
        REF  VMBW,VMBR.VWTR
        REF  XMLLNK.NUMREF
*
CFI     DATA >12B8
ROOM    BSS  >C0
*
MODE    LWPI MYREG
        LI   R1,1
        BL   @NIMNUM
*
MODE1   CI   R5,>0001
        JNE  MODE2
        BL   @MOHELP
        BLWP @VMBW
        LI   R0,>E001
        MOVB R0,@>83D4
        SWPB R0
        BLWP @VWTR
        LI   R0,>030C
        BLWP @VWTR
        LI   R0,>0717
        BLWP @VWTR
        LI   R1,>8000
        BL   @CLT
*
```

```
*
MODE2  CI   R5,>0002
       JNE  MOUT
       BL   @MOHELP
       BLWP @VMBR
       LI   R0,>F001
       MOVB R0,@>83D4
       SWPB R0
       BLWP @VWTR
       LI   R0,>0320
       BLWP @VWTR
       LI   R0,>07F4
       BLWP @VWTR
       LI   R1,>8000
       BL   @CLT
*
MOUT   CLR  R0
       MOVB R0,@>837C
       LWPI >83E0
       B    @>0070
*
MOHELP LI   R0,>0301
       LI   R1,ROOM
       LI   R2,>00C0
       RT
*
CLT    BLWP VSBW
       INC  R0
       C    R0,960
       JLE  CLT
       RT
*
NIMNUM BLWP @NUMREF
       BLWP @XMLLNK
       DATA CFI
       MOV  @FAC,R5
       RT
*
       END
```

VDPEEK und VDPOKE

Mit dem Extended Basic Modul haben Sie keinen direkten VDP-Zutritt. Die Befehle PEEKV und POKEV, welche im E/A-Modul integriert sind, fehlen. Die folgende Routine mit den beiden Eintrittsstellen VDPEEK und VDPOKE bietet Ersatz. Im Gegensatz zu den Routinen PEEKV und POKEV des E/A-Moduls und den Originalbefehlen unserer Diskette 'TORPEDO BASIC' erlauben die beiden Maschinenbefehle, die nachfolgend abgedruckt sind, nur einen Parameterwert und enthalten auch keine Fehlerkontrolle.

Mit der Instruktion

CALL LINK("VDPEEK",Adresse,numerische Variable)

können Sie einen Wert aus dem VDP-Ram lesen.

Mit der Instruktion

CALL LINK("VDPOKE",Adresse,Wert oder Ausdruck)

können Sie einer VDP-Adresse einen Wert zuweisen.

Die Adresse muß zwischen 0 und 16383 sein (>0 - >3FFF).

Die Routinen VDPEEK und VDPOKE können nicht mit dem Basic des E/A-Moduls verwendet werden, es sei denn, Sie ändern die Zutrittsadressen der verschiedenen benutzten Hilfsroutinen.

Beachten Sie, daß durch poken bestimmter Werte im VDP-Ram schwerwiegende Veränderungen erfolgen können, welche ein Aus- und Wiedereinschalten des Computers erforderlich machen können. Speichern Sie daher Programme, welche diese Routinen ansprechen, vor der ersten Benutzung unbedingt ab!

```
       DEF  VDPEEK,VDPOKE
*
VSBW   EQU  >2020
VSBR   EQU  >2028
XMLLNK EQU  >2018
NUMREF EQU  >200C
NUMASG EQU  >2008
*
CIF    DATA >20
CFI    DATA >1288
FAC    EQU  >834A
*
VDPOKE LWPI MYREG
       LI   R1,1
       BL   @NIMNUM
*
       MOV  R5,R6
       LI   R1,2
       BL   @NIMNUM
*
       MOV  R6,R0
       MOV  R5,R1
       SWPB R1
       BLWP @VSBW
       JMP  OUT
*
VDPEEK LI   R1,1
       BL   @NIMNUM
       MOV  R5,R0
       BLWP @VSBR
       MOV  R1,R5
       LI   R1,2
       BL   @GIBNUM
       JMP  OUT
*
OUT    CLR  R0
       MOVB R0,@>837C
       LWPI >83E0
       B    @>0070
*
NIMNUM CLR  R0
       BLWP @NUMREF
       BLWP @XMLLNK
       DATA CFI
       MOV  @FAC,R5
       RT
*
GIBNUM CLR  R0
       MOV  R5,@FAC
       BLWP @XMLLNK
       DATA CIF
       BLWP @NUMASG
       RT
*
       END
```

4

4. ANWENDUNGSHINWEISE

In Verbindung mit dem Assembler Kurs II haben Sie jetzt bereits ein Umfangreiches Wissen über die Maschinensprache des TI-99/4a. Bicher wird sich der ein oder andere jetzt die Frage nach den Verwendungsmöglichkeiten stellen.

Als bestes Beispiel hierfür mögen unsere Systemerweiterungen dienen, die nur mit Maschinensprache möglich sind. Nur für Spiele zu programmieren ist Assembler sicherlich zu schade - obwohl natürlich alle professionellen Spiele (vor allem der Schnelligkeit wegen) in Maschinensprache programmiert sind.

Einen ganz großen Vorteil hat die Maschinensprache zweifellos in der Datenverwaltung und Textbearbeitung. Zwar sind wir in diesem Band noch nicht auf das Ansprechen von Peripheriegeräten eingegangen; trotzdem sollte es Ihnen möglich sein, vielleicht sogar unter Verwendung der verschiedenen Modi Text-, Verwaltungs-, Statistik- und Dateiprogramme zu schreiben. Selbst ein kleines Zeichenprogramm im Bit-Map-Modus sollte Ihnen keine Probleme mehr bereiten.

Unser Vertrieb ist übrigens immer auf der Suche nach guten Programmen in Maschinensprache. Wenn Sie etwas derartiges geschrieben haben, lassen Sie es uns doch ruhig einmal testen. Wenn wir uns für einen Vertrieb entscheiden, winkt sogar ein Honorar (Allen Einsendungen aber bitte eine ausführliche Dokumentation/Erläuterung und einen Freiumschlag für eine evtl. Rücksendung beifügen).

5

5. TABELLEN UND LISTEN

In diesem Kapitel möchten wir Ihnen noch einige Tabellen und Listen zur Verfügung stellen, welche Ihnen sicherlich bei der Erstellung eigener Programme von Nutzen sein werden.

Weitere Tabellen finden Sie in Ihrem Handbuch zum EDITOR/ASSEMBLER (soviel Englisch wird wohl jeder verstehen, aber es gibt dieses Manual übrigens auch in Deutsch).

MNEMONIC-ÜBERSICHT

In der folgenden Übersicht finden die nachstehend aufgeführten Buchstaben zur Kennzeichnung verwendung:

A: Befehlsname
B: Maschinencode
C: Befehlsformat
D: Kurzbeschreibung
E: Status-Register-Beeinflussung
(Wenn nur Test, Bits in Klammern)
\- Keine angabe

Eine Syntaxbeschreibung finden Sie im ASSEMBLER KURS II sowie im Handbuch zum Editor/Assembler.

A: A
B: A000
C: Worte addieren
D: I
E: L> A> EQ C OV

A: AB
B: B000
C: Bytes addieren
D: I
E: L> A> EQ C OV

A: ABS
B: 0740
C: Absolutwert
D: VI
E: L> A> EQ OV

A: AI
B: 0220
C: Unmittelbar addieren
D: VIII
E: L> A> EQ C OV

A: ANDI
B: 0240
C: Unmittelbare UND-Verknüpfung
D: VIII
E: L> A> EQ

A: B
B: 0440
C: Verzweigung
D: VI
E: keine

A: BL
B: 0680
C: Sprung und Verknüpfung
D: VI
E: keine

A: BLWP
B: 0400
C: Sprung und Registerzeiger laden
D: VI
E: keine

A: C
B: 8000
C: Worte vergleichen
D: I
E: L> A> EQ

A: CB
B: 9000
C: Bytes vergleichen
D: I
E: L> A> EQ OP

A: CI
B: 0280
C: Unmittelbarer Vergleich
D: VIII
E: L> A> EQ

A: CKOF
B: 03C0
C: Interne Zeittakt-Abschaltung
D: VII
E: -

A: CKON
B: 03A0
C: Interne Zeittakt-Einschaltung
D: VII
E: -

A: CLR
B: 04C0
C: Worte löschen (mit '0' füllen)
D: VI
E: keine

A: COC
B: 2000
C: Korrespondierende Einsen vergleichen
D: III
E: EQ

A: CZC
B: 2400
C: Korrespondierende Nullen vergleichen
D: III
E: EQ

A: DEC
B: 0600
C: Decrementieren
D: VI
E: L> A> EQ C OV

A: DECT
B: 0640
C: Decrementieren in Zweierschritten
D: VI
E: L> A> EQ C OV

A: DIV
B: 3C00
C: Dividieren
D: IX
E: OV

A: IDLE
B: 0340
C: Zeittakt-Leerlauf
D: VII
E: -

A: INC
B: 0580
C: Incrementieren
D: VI
E: L> A> EQ C OV

A: INCT
B: 05C0
C: Incrementieren in Zweierschritten
D: VI
E: L> A> EQ C OV

A: INV
B: 0540
C: Invertieren
D: VI
E: L> A> EQ

A: JEQ
B: 1300
C: Sprung wenn EQ gesetzt
D: II
E: (EQ)

A: JGT
B: 1500
C: Sprung wenn A gesetzt
D: II
E: (A)

A: JH
B: 1800
C: Sprung wenn L> gesetzt und EQ zurückgesetzt
D: II
E: (L> EQ)

A: JHE
B: 1400
C: Sprung wenn L> oder EQ gesetzt
D: II
E: (L> EQ)

A: JLE
B: 1200
C: Sprung wenn EQ gesetzt und L> zurückgesetzt
D: II
E: (L> EQ)

A: JLT
B: 1100
C: Sprung wenn A> und EQ zurückgesetzt
D: II
E: (A> EQ)

A: JMP
B: 1000
C: Unbedingter Sprung
D: II
E: keine

A: JNC
B: 1700
C: Sprung wenn C zurückgesetzt
D: II
E: (C)

A: JNE
B: 1600
C: Sprung wenn EQ zurückgesetzt
D: II
E: (EQ)

A: JNO
B: 1900
C: Sprung wenn OV zurückgesetzt
D: II
E: (OV)

A: JOC
B: 1800
C: Sprung wenn C gesetzt
D: II
E: (C)

A: JOP
B: 1C00
C: Sprung wenn OP gesetzt
D: II
E: (OP)

A: LDCR
B: 3000
C: CRU laden
D: IV
E: L> A> EQ OP

A: LI
B: 0200
C: Register unmittelbar laden
D: VIII
E: L> A> EQ

A: LIMI
B: 0300
C: Interrupt Maske unmittelbar laden
D: VIII
E: INT. MASK

A: LREX
B: 03E0
C: Laden oder Durchführung neu starten
D: VII
E: -

A: LWPI
B: 02E0
C: Registerbereichszeiger unmittelbar laden
D: VIII
E: keine

A: MOV
B: C000
C: Worte verschieben (kopieren)
D: I
E: L> A> EQ

A: MOVB
B: D000
C: Bytes verschieben (kopieren)
D: I
E: L> A> EQ OP

A: MPY
B: 3800
C: Multiplizieren
D: IX
E: keine

A: NEG
B: 0500
C: Negieren
D: VI
E: L> A> EQ OV

A: ORI
B: 0260
C: Unmittelbare ODER-Verknüpfung
D: VIII
E: L> A> EQ

A: RSET
B: 0360
C: Uhrtakt-Zurücksetzung
D: VII
E: -

A: RTWP
B: 0380
C: Rücksprung mit Registerbereichszeiger
D: VII
E: L> A> EQ C OV OP X INT. MASK

A: S
B: 6000
C: Worte subtrahieren
D: I
E: L> A> EQ C OV

A: SB
B: 7000
C: Bytes subtrahieren
D: I
E: L> A> EQ C OV

A: SBO
B: 1D00
C: CRU-Bit auf '1' setzen
D: II
E: keine

A: SBZ
B: 1E00
C: CRU-Bit auf '0' setzen
D: II
E: keine

A: SETO
B: 0700
C: 16-Bit-Wort auf '1..1' setzen (>FFFF)
D: VI
E: keine

A: SLA
B: 0A00
C: Arithmetische Linksverschiebung
D: V
E: L> A> EQ C OV

A: SOC
B: E000
C: Korresoondierende Einsen setzen
D: I
E: L> A> EQ

A: SOCB
B: F000
C: Korrespondierende Einsen im hochwertigen Byte setzen
D: I
E: L> A> EQ OP

A: SRA
B: 0800
C: Arithmetische Rechtsverschiebung
D: V
E: L> A> EQ C

A: SRC
B: 0B00
C: Zyklische Rechtsverschiebung (Rechtsrotation)
D: V
E: L> A> EQ C

A: SRL
B: 0900
C: Logische Rechtsverschiebung
D: V
E: L> A> EQ C

A: STCR
B: 3400
C: CRU abspeichern
D: IV
E: L> A> EQ OP

A: STST
B: 02C0
C: Status in Register kopieren
D: VIII
E: keine

A: STWP
B: 02A0
C: Registerbereichszeiger in Register kopieren
D: VIII
E: keine

A: SWPB
B: 06C0
C: Bytes im Wort vertauschen
D: VI
E: keine

A: SZC
B: 4000
C: Korrespondierende Nullen setzen
D: I
E: L> A> EQ

A: SZCB
B: 5000
C: Korrespondierende Nullen im hochwertigen Byte setzen
D: I
E: L> A> EQ OP

A: TB
B: 1F00
C: CRU-Bit testen
D: II
E: EQ

A: XOR
B: 2800
C: Exclusive ODER-Verknüpfung
D: III
E: L> A> EQ

Nicht berücksichtigt: X und XOP.

REGISTER UND VDP-REGISTER

R0 - R2 Werden durch eingebaute Utilities benutzt.
R3 - R10 Frei für eigene Verwendung.
R11 Enthält Rücksprungadresse nach BL.
R12 Enthält in den Bits 3-14 die CRU-Basis Adresse.
R13 Enthält den Arbeitsregister Pointer nach BLWP.
R14 Enthält den Programm-Zeiger nach BLWP.
R15 Enthält das Statusregister nach BLWP.

VDP-R0 Initialisiert durch Bit 6 den Bit MAP Modus; ermöglicht oder unterdrückt mit Bit 7 die Video-unterbrechung.

VDP-R1 Initialisiert in Bit 3 den Text Modus, in Bit 4 den Multicolormodus; in den Bits 6 und 7 erfolgt die Sprite-Größenauswahl.

VDP-R2 Startadresse der Bildschirmdarstellungstabelle (Inhalt des Registers multipliziert mit >400.

VDP-R3 Startadresse der Farbtabelle (Inhalt des Registers multipliziert mit >40).

VDP-R4 Startadresse der Musterbeschreibungstabelle (Inhalt des Registers multipliziert mit >800).

VDP-R5 Startadresse der Sprite-Attributen-Tabelle (Inhalt des Registers multipliziert mit >80).

VDP-R6 Startadresse der Spritemusterbeschreibungstabelle (Inhalt des Registers multipliziert mit >800).

VDP-R7 Enthält in den niedrigwertigen 4 Bits die Bildschirm-farbe und im Text-Modus in den 4 hochwertigen Bits die Characterfarbe.

FARBTABELLE

0 Transparent	8 Mittelrot
1 Schwarz	9 Hellrot
2 Mittelgrün	10 A Dunkelgelb
3 Hellgrün	11 B Hellgelb
4 Dunkelblau	12 C Dunkelgrün
5 Hellblau	13 D Magentarot
6 Dunkelrot	14 E Grau
7 Kornblumenblau	15 F Weiß

EXTENDED BASIC GLEICHSTELLUNGEN

```
FAC     EQU >834A
GPLWS   EQU >83E0
GRMRD   EQU >9800
GRMRA   EQU >9802
GRMWD   EQU >9C00
GRMWA   EQU >9C02
PAD     EQU >8300
SCAN    EQU >000E
SOUND   EQU >8400
SPCHWT  EQU >9400
VDPRD   EQU >8800
VDPSTA  EQU >8802
VDPWA   EQU >8C02
VDPWD   EQU >8C00
```

UTILITY-ROUTINEN IN EXTENDED BASIC

```
ASSGNV EQU >18
CFI    EQU >12B8
CIF    EQU >20
CNS    EQU >06
COMPCT EQU >00
CSN    EQU >11AE
ERR    EQU >2034
FADD   EQU >0D80
FCOMP  EQU >0D3A
FDIV   EQU >0FF4
FMUL   EQU >0E88
FSUB   EQU >0D7C
GETSTR EQU >02
GVWITE EQU >36
KSCAN  EQU >201C
MEMCHK EQU >04
NEXT   EQU >0070
NUMASG EQU >2008
NUMREF EQU >200C
SADD   EQU >0D84
SCROLL EQU >26
SDIV   EQU >0FF8
SMUL   EQU >0E8C
SSUB   EQU >0D74
STRASG EQU >2010
STRREF EQU >2014
VGWITE EQU >34
VMBR   EQU >202C
VMBW   EQU >2024
VPOP   EQU >10
VPUSH  EQU >0E
VSBR   EQU >2028
VSBW   EQU >2020
VWTR   EQU >2030
XMLLNK EQU >2018
```

FEHLERMELDUNGS-VERKETTUNG

>02 Numeric overflow (Kapazitätsüberlauf numerisch)
>03 Syntax error (Fehlerhaftes Befehlsformat)
>04 Illegal after subprogram (Nach Unterprogramm unzulässig)
>05 Unmatched quotes (Anführungszeichen falsch gesetzt)
>06 Name too long (Zu langer Variablenname)
>07 String number mismatxh (Falsche Zuordnung String/Zahl)
>08 Option Base Error (Falsche Dimensionsbasis/Feldelemente)
>09 Improperly used name (Bezeichnung nicht zulässig)
>0A Image error (Maskierungsfehler bei Displayausgabe)
>0B Memory full (Speicher belegt)
>0C Stack overflow (Stack-Speicher-überlauf)
>0D Next without For (Schleife beendet, ohne zu öffnen)
>0E For-Next nesting (Schleife geöffnet, ohne zu beenden)
>0F Must be in subprogram (Muß im Unterprogramm enthalten sein)
>10 Recursive subprogram call (rekursiver Unterprogrammaufruf)
>11 Missing subend (SUBEND-Befehl nicht vorhanden)
>12 Return without Gosub (Rücksprunganweisung ohne Hinsprung)
>13 String truncated (Zeichenkette wurde verkürzt)
>14 Bad subscript (Falscher Index)
>15 Speech string too long (Phonetischer String zu lang)
>16 Line not found (Zeilennummer nicht gefunden)
>17 Bad line number (Unzulässige Zeilennummernbezeichnung)
>18 Line too long (Zeile zu lang)
>19 Cant continue (Fortsetzung nicht möglich)
>1A Command illegal in program (Direktbefehl falsch im Programm)
>1B Only legal in a program (Programmbefehl im Direktmodus)
>1C Bad Argument (falsches Argument (zu groß/klein)
>1D No program present (Kein Programm vorhanden)
>1E Bad Value (Falscher numerischer Wert)
>1F Incorrect argument list (Zu wenig oder zuviele Argumente)
>20 Input error (Eingabefehler Zahl/String)
>21 Data error (Read liest über vorhandene Daten hinaus)
>22 File error (Dateifehler)
>25 I/O error (Ein-/Ausgabebefehl falsch)
>27 Protection violation (Programmschutzverletzung)
>28 Unrecognized Character (Unerkannter Character)

WARNUNGSMELDUNG-VERKETTUNG

>29 Numeric overflow (Kapazitätsüberlauf numerisch)
>2A String truncated (Zeichenkette abgeschnitten)
>2B No program present (Kein Programm vorhanden)
>2C Input error (Eingabefehler)
>2D I/O error (Ein-/Ausgabebefehl falsch)

RADIX 100 SCHREIBWEISE

(Das Ausdrücken einer Zahl als 100er-Potenz)

Beispiele:

1040 = 0.1040 x 100 hoch 2.
35 = 35 x 100 hoch 0.
104.7 = 1.047 x 100 hoch 1.

In dieser Form werden numerische Parameter vom Maschinenprogramm ans Basic und umgekehrt übergeben.

BENUTZUNG DER SPEICHERERWEITERUNG

Editor Assembler Modul:

```
>2000 - >2001 ID-Code >A55A
>2002 - >2021 XML-Vektor benutzt vom E/A
>2022 - >20FF UTLTAB Utility-Datenbereich
>2100 - >2127 Utility BLWP-Vektoren
>2128 - >26FF Utility-Routinen
>2700 - >3F37 Assembler Programme
>3F38 - >3FFF REF/DEF-Tabelle

>A000 - >FFE0 Assembler Programme
```

Extended Basic Modul:

```
>2000 - >2001 XML-Vektor benutzt vom E/A
>2002 - >2005 Utility-Datenbereich
>2006 - >2007 ID-Code >AA55
>2008 -       Utility-BLWP-Vektoren
              Utility-Routinen
              Assembler Programme
      - >2FFF DEF-Tabelle

>A000 -       Freier Platz und CPU-RAM Endadresse >8386
              Numerischer Wertbereich
              Zeilennummerntabelel
        >FFE0 Extended Basic Programme
```

VDP-RAM BENUTZUNG IN DEN VERSCHIEDENEN MODI

0 = Freier oder reservierter Raum
1 = Bildschirmdarstellungstabelle
2 = Musterbeschreibungstabelle
3 = Farbtabelle
4 = Sprite-Attributen-Liste
5 = Sprite-Musterbeschreibungstabelle
6 = Sprite-Bewegungstabelle
. = Bereich geht über diese Stelle hinaus
_ = Bereich endet hier

Speicherbereich	Grafik	Text	Multicolor	Bitmap
>0000	1	1	1	2/3
- >02FF	_	.	_	.
>0300	4	.	4	.
- >037F	_	.	_	.
>0380	3	.	0	.
- >03BF	.	_	.	.
>03C0	.	0	.	.
- >03FF	_	.	_	.
>0400	5	.	2	.
- >077F	_	.	.	.
>0780	6	.	.	.
- >07FF	_	_	.	.
>0800	2	2	.	.
- >09FF	_	_	_	.
>1000	0	0	0 Pabs	.
- >17FF	.	.	. Sprites	_
>1800	.	.	. <_____/	3/2
- >2FFF	.	.	.	_
>3000	.	.	. Sprites	0
- >37D6	_	_	_ L_____>	_
>37D7	0	0	0	0
- >3FFF	_	_	_	_

ASSEMBLER KURS III (C) HAGERA

TORPEDO BASIC EXPANSION

WAS IST DAS EIGENTLICH?

TORPEDO ‚BASIC ist eine Systemerweiterung, welche viele Möglichkeiten heute aktueller Computer endlich auch auf dem TI-99/4a verfügbar macht.

Pluspunkt 1: Neben dem Grafik-Modus wird der Text-Modus des TI-99/4a bereitgestellt. In diesem Modus, welcher bislang vielen Benutzern des TI überhaupt nicht bekannt ist, werden in 24 Zeilen jeweils 40 Zeichen ausgegeben. Vor allem Verwaltungsprogramme profitieren von diesem Modus - eine Textbearbeitung zum Beispiel ist mit weniger als 40 Zeichen/Zeile undenkbar.

Pluspunkt 2: Die Erweiterung stellt einen speziellen RAM-Buffer zur Verfügung, welcher exakt die jeweilige Bildschirmgröße hat. So können ganze Bildschirme blitzschnell versteckt, ausgetauscht und wieder herbeigeholt werden.

Pluspunkt 3: TORPEDO BASIC verfügt über eine hervorragende Window-Technik. Bis zu 32 Bildschirmfenster können unabhängig voneinander definiert werden, sowohl mit Zeilen- als auch mit Spaltenbegrenzung. Die Windows können natürlich in beiden Modi verwendet werden.

Pluspunkt 4: Mit der implementierten Hardcopy-Routine können Sie ganze Bildschirme oder Ausschnitte daraus zu Papier bringen. Die Routine funktioniert auf allen Druckern mit Epson-kompatiblen Steuerzeichen (das sind die meißten Drucker).

Pluspunkt 5: Zum Auslisten des Disketten-Kataloges können Sie ab Sofort auf Modulwechsel verzichten. Auch hierfür wird eine Maschinenroutine zur Verfügung gestellt. Die Bildschirmausgabe kann dabei auf einen bestimmten Ausschnitt begrenzt werden, um Bildschirminhalte nicht zu zerstören.

Pluspunkt 6: Endgültig Schluß ist auch mit umständlichen IF-Abfragen oder dem in seiner Wirkung arg begrenzten ON-GOTO Befehl. Mit TORPEDO BASIC können Sie zu einer Zeilennummer 'x' verzweigen, wobei 'x' das Ergebnis eines jeden numerischen Ausdrucks oder eine Variable mit Werten zwischen 1 und 32767 sein kann.

Pluspunkt 7: Eingebaut ist weiterhin eine Interruptroutine, welche es ermöglicht, 9 internationale Zeichensätze und einen wissenschaftlichen Zeichensatz zur Verfügung zu stellen sowie auch im Direkt-Modus mit beliebigen Bildschirm- und Schriftfarben zu arbeiten. Auch der Befehl CALL CLEAR im Direktmodus kann getrost vergessen werden - <CTRL>+<1> hat in TORPEDO BASIC dieselbe Funktion!

Pluspunkt 8: Endlich besitzt auch der TI-99/4a die hervorragenden Eigenschaften des bildschirmorientierten Cursors. Damit sind Sie mit direkten Bildschirmeingaben nicht mehr auf eine Zeile beschränkt. Alle Fenster können mit diesem Cursor benutzt werden, welcher alle nötigen Funktionen zur Verfügung stellt: Delete + Insert Character, Erase Line, Begin (Home), Clear Window, Back, Enter und alle vier Steuerrichtungen.

Pluspunkt 9: Für Extended Basic werden außerdem die Befehle VDPEEK und VDPOKE zur Verfügung gestellt. Damit ist direkter Zugriff auf das VDP-Ram möglich. Diese Befehle sind im E/A-Modul bereits vorhanden. Was Extended Basic Besitzern bisher leider nicht möglich war, schafft TORPEDO BASIC.

DIE NEUEN BEFEHLE IN DER ÜBERSICHT

BRANCH	Verzweigung zu einer Zeilennummer 'x'.
CHRSET	Laden eines Sonderzeichensatzes.
CLEAN	Löschen von 1-255 Charactern (Zeile) oder Überschreiben mit einem ASCII-Code in beliebiger Richtung.
CLTEXT	Löschen des Bildschirms oder von Fenstern oder Überschreiben mit einem ASCII-Code.
COPY	Hardcopy vom Bildschirm oder von Bildschirmfenstern auf jedem Drucker mit Epson-kompatiblen Steuercodes.
DIR	Ausgabe des Diskettenkataloges auf einem begrenzten Bildschirmraum.
INT	Eingaben auf dem Bildschirm mit optionaler Datentypbegrenzung, festlegbarem Format etc.

GETSTR	Übernahme eines Strings vom Screen in eine Variable.
HIDE	Kopieren eines Fensters in den Buffer.
INSTR	Schreiben eines Strings in den Buffer.
MODE	Umschalten zwischen Grafik- und Textmodus.
OUTSTR	Übernahme eines Bufferbereichs in eine Variable.
QUIT	Aufruf des TI-Titelbildes aus dem Programm.
SCROLL	Bewegen des Bildschirms oder von Fenstern in eine beliebige Richtung und Füllen freiwerdender Stellen mit beliebigen Zeichen.
SEARCH	Auffinden eines ASCII-Codes oder Strings in einem Fenster oder auf dem gesamten Bildschirm.
SWAP	Austausch des Bildschirms oder eines Fensters mit dem Bufferinhalt.
TABLE	Arbeiten mit dem bildschirmorientiertem Cursor.
TAKE	Beschreiben des Bildschirms oder eines Fensters mit dem Bufferinhalt.
TORPED	Initialisierung der Systemerweiterung.
USECOL	Festlegung der Bildschirmfarben sowie Starten und Stoppen der Interruptroutine.
VDPEEK*	Lesen von Bytes aus dem VDP-Ram.
VDPOKE*	Schreiben von Bytes in das VDP-Ram.
WINDOW	Definieren eines Bildschirmfensters.
WRITE	Ausgabe von Zahlen und Strings auf dem Bildschirm.

(* Auf der E/A-Version nicht vorhanden, da im Modul unter dem Namen PEEKV/POKEV eingebaut.)

WEITERE VORTEILE VON TORPEDO BASIC

Dies ist natürlich noch lange nicht alles. So ganz 'nebenbei' kann man jetzt den Bildschirm oder Teile daraus in jede Richtung scrollen und freiwerdenden Raum mit beliebigen Zeichen füllen, Das TI-Titelbild aus dem Programm heraus aufrufen, Zeichen und Strings blitzschnell auffinden und Funktionen wie HCHAR und VCHAR in jede Richtung (nicht nur nach rechts und unten) darstellen.

TORPEDO BASIC macht Ihren 'alten' TI-99/4a wieder flott. Wenn auch Sie sehen möchten, wie Besitzer populärer Homecomputer, die den TI für'n Appel und'n Ei verschleudert haben, vor Neid erblassen, sollten Sie sich diese Systemerweiterung nicht entgehen lassen.

LIEFERUMFANG UND PREISE

Das Programm-Paket TORPEDO BASIC enthält:

1 Datenträger mit dem Maschinenprogramm;
1 ausführliches Bedienungshandbuch, 104 Seiten.

Lieferbar ist das Paket für folgende Konfigurationen:

Art.-Nr. 04311	TORPEDO BASIC für Editor/Assembler-Modul Diskette und Handbuch komplett	DM	99.00
Art.-Nr. 04312	TORPEDO BASIC für Extended-Basic - Modul Cassette mit Basiclader, MC-Code + Handbuch komplett	DM	129.00
Art.-Nr. 04316	TORPEDO BASIC für Extended-Basic - Modul Diskette und Handbuch komplett	DM	99.00
Art.-Nr. 04319	TORPEDO BASIC für Editor/Assembler Modul und Extended Basic Modul komplett zusammen auf einer Diskette + Handbuch	DM	129.00
Art.-Nr. 08091	TORPEDO BASIC Handbuch Vorablieferung	DM	24.90

NACHWORT ZUM ASSEMBLER KURS III

Wenn Sie auf dieser Seite angelangt sind, können Sie durchaus von sich behaupten, die Assemblersprache des TI-99/4a zu beherrschen. Sicher werden noch die ein oder anderen Fragen offen sein, welche dieses Buch aber nicht beantworten kann oder soll.

Ich habe mich bemüht, den Text so leichtverständlich wie möglich zu halten. Sollten Sie etwas nicht verstanden haben, schreiben Sie bitte an die Anschrift des Vertriebes, und nicht an meine Privatadresse. Ihre Post kann dann schneller beantwortet werden.

Über jegliche Kritik zu diesem Band würde ich mich sehr freuen, auch wenn Sie nur Gutes zu berichten haben.

Bitte schreiben Sie auch, wenn Ihrerseits Interesse an weiteren Bänden besteht. Dieser Kurs III verdankt seine Existenz einzig und allein dem positiven Echo auf Kurs II und den vermehrten Bitten um eine Fortsetzung. Stoff ist genügend vorhanden - und wenn Sie sich selbst beteiligen wollen (gegen Honorar, versteht sich), würde ich mich über eine entsprechende Einsendung sehr freuen.

Dabei denke ich hauptsächlich an einen 'PROFESSIONAL'-Kurs und an eine 'ASSEMBLER-PROGRAMM-SAMMLUNG'.

In Erwartung Ihrer Antwort verbleiben ich
und der Vertrieb

Mit freundlichen Grüßen

Hans-Georg Rausch
Autor und
verantw. Gesellschafter

- Thanks to 99'er User: ***Alfredo Cevolini***, for the Scan of this document.

- Reworked by TI99 Italian User Club (info@ti99iuc.it) - 2014

Hagera