65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DM 7,80 Nr. 16 Dezember 1980

Den gründlichen Darstellungen

wurde in diesem Heft wiederum ausreichend Platz eingeräumt, denn nur mit genügend Erklärungen und Kommentaren kann dem Anwender der Nachvollzug der Programmierung nahegebracht werden. Die Beiträge sollen ja mit ihren wertvollen Gedanken nicht allein stehen. Sie sollen im Sinne eines Softwareforums zu eigener Arbeit an ähnlichen Aufgabestellungen anregen. Immer häufiger sehen wir die Fortentwicklung früherer Arbeiten zum Nutzen für alle Leser, die Autoren einbegriffen.

Bei der Fülle vorliegender Themen mußten einzelne Beiträge wiederum in eine Serie gesplittet werden, andere können erst in den Folgeheften erscheinen. Die Autoren werden um Verständnis gebeten, ihnen sei am Ende des Jahres für den fruchtbaren Gedankenaustausch gedankt.

Dieses Jahr endet nach entsprechenden Anstrengungen zugleich mit einer erheblichen Verbesserung der graphischen Aufmachung der Hefte, von der der Herausgeber hofft, daß sie dieser wegen ihres Inhaltes geschätzten Fachzeitschrift auch auf diese Weise neue Freunde gewinnt.

Inhaltsverzeichnis

Schreiben Sie lieber deutsch als Assembler?

Wenn Sie sich dann noch für kleine und kleinste Computersysteme interessieren, könnten sie unser Mann (unsere Frau) sein:
Wir suchen einen weiteren Mitarbeiter für den Bereich Computer und Peripherie in unserer Redaktion.

Zu den Aufgabengebieten eines Markt & Technik-Fachredakteurs gehören unter anderem:

- Gespräche, Interviews mit führenden Persönlichkeiten der Elektronikbranche.
- Der Besuch von in- und ausländischen Messen, Pressekonferenzen und Unternehmen.
- Das Bearbeiten der Berichte über neue Produkte und Technologien, die laufend von allen wichtigen Firmen der Branche geschickt werden (dadurch bleibt das Wissen des Redakteurs immer auf dem allerneuesten Stand und er erhält außerdem einen einzigartigen Marktüberblick in seinem Fachgebiet).
- Recherchen zu aktuellen Fragen oder zu speziellen Themen — beispielsweise zur Preisentwicklung oder den Nutzen von Standard-Software.

Sie müssen Kenntnisse auf dem Gebiet der Datenverarbeitung mitbringen. Wir können Ihnen eine interessante, abwechslungsreiche Tätigkeit bieten und bezahlen leistungsgerecht. Sie können (und sollen) selbständig arbeiten.

Eintrittstermin: spätestens 1.4.81.

Wir informieren Sie gern über die Möglichkeiten, die wir Ihnen bieten können. Zur Kontaktaufnahme genügt ein kurzer Brief oder ein Anruf (Tel. 089/3006061) beim zuständigen Ressortredakteur S. Fiedler.

Markt&Technik

Verlagsgesellschaft mbH, Winzererstraße 106, 8000 München 40

Dipl.-Math. A. Quint, Wiesbaden

Ein Sortierprogramm für CBM 3001

Sortieren ist ein in der Datenverarbeitung häufiger und zeitaufwendiger Vorgang. Sortiert werden muß, wenn Listen z.B. nach Namen sortiert ausgedrückt werden müssen. Sortieren empfiehlt sich, wenn damit der Zugriff auf einzelne Informationen beschleunigt werden kann, indem man ein binäres Suchen anwendet. - Sortiert werden muß, wenn man bei einer Direktzugriffsdatei nach mehreren (z.B. 3) verschiedenen Kriterien zugreifen will und dabei innerhalb von 4 Sekunden die Antwort auf dem Schirm haben will. Da BASIC-Programme sehr viel Zeit benötigen, lohnt es sich, eine Assemblerroutine für diesen Zweck zu schreiben.

Das folgende Sortierprogramm soll nicht ein von einem anderen Computer übernommenes Programm sein, das auf den cbm 3001 gezwungen wurde. In einem solchen Fall müßten erst die Strings zu einem kompakten Block zusammengeschoben werden, damit die Voraussetzungen erfüllt sind, die in anderen Computersystemen mit einer anderen Speicherverwaltung gegeben sind. Vielmehr soll die Darstellung der Strings im cbm 3001 unverändert übernommen werden. In diesem Falle kann man Strings durch Vertauschen der Pointer (Startadresse und Länge) umspeichern, ohne die Speicherung der eigentlichen Zeichenfolge zu verändern. Bei längeren Strings (Länge größer 3 Bytes) ist dies natürlich von Vorteil. Außerdem müssen die zu sortierenden Strings nicht dieselbe Länge haben, was beim Umspeichern der Zeichenfolgen Voraussetzung ist, oder man erzwingt die zeitaufwendige Garbage-Collection-Routine.

Um die Darstellung der Strings im cbm auszunutzen, muß man wissen, wo sie gespeichert werden. Im 65xx MICRO MAG Nr. 14 wurde dargstellt, wie Variable verschlüsselt und wie Strings gespeichert werden. Für Felder gilt dies analog, nur müssen zusätzliche Informationen gespeichert werden. Das 1. und 2. Bytes eines Feldes enthält wieder den Namen. Byte 3 und 4 enthält die Anzahl der insgesamt benötigten Bytes für dieses Feld (LO, HI), so daß man daraus den Beginn des nächsten Feldes errechnen kann. Byte 5 enthält die Anzahl der Dimensionen dieses Feldes (d.h. wieviele Ebenen das Feld hat). Die folgenden Bytes enthalten die Anzahl der Elemente in einer Dimension (Achtung: Nicht höchster Index, sondern Anzahl!). Beispiel:

```
DIM AB%(5) WIRD ZU
$C1,C2,13,00,01,00,06
```

in den folgenden Bytes stehen die jeweiligen Werte von AB%(.).

Bei mehreren Dimensionen (Byte Nr. 5 $\neq$ 01) stehen in den Bytes 6 und 7 (und bei Bedarf 8 und 9, 10 und 11 usw.) die Anzahl der Elemente in dieser Dimension. Dabei ist die Reihenfolge vertauscht, das Byte 6 und 6 enthält die Anzahl der Elemente in der letzten angegebenen Dimension. Beispiel:

```
DIM A%(3,2,1) WIRD ZU
$C1,80,3B,00,03,00,02,00,03,00,04, ...
```

die folgenden Bytes enthalten die Werte von A%(.,.,.), und zwar

```
DAS 1. BYTEPAAR ENTHÄLT A%(0,0,0)
    2. "        "       A%(1,0,0)
    3. "        "       A%(2,0,0)
    4. "        "       A%(3,0,0)
    5. "        "       A%(0,1,0) USW.
```

Beschränken kann man sich auf Stringfelder, da im allgemeinen Strings sortiert werden müssen.

Der Sortieralgorithmus

Das Verfahren nennt sich Auswahlverfahren mit Austausch. Es funktioniert folgendermaßen: Aus der Liste der Strings sucht man den alphabetisch kleinsten String. Dieser tauscht seinen Platz mit A $(0). Ab A $(1) sucht man jetzt den kleinsten String, der natürlich nur der zweitkleinste String von A $(0) - A$(N) sein kann. Dieser tauscht seinen Platz mit A $(1) usw.. Ein BASIC-Programm des Algorithmus:

```
 10 N=100:DIMA(N):X=RND(-1)
 20 FORI=ØTON:A(I)=INT(1ØØØ*RND(1)):NEXT
 30 PRINT"BEGINN DER SORTIERUNG"
100 T1=TI
120 FORI=ØTON-1:K=I
130 FORJ=ITON
150 IFA(J)<A(K)THENK=J
160 NEXTJ
170 X=A(I):A(I)=A(K):A(K)=X
180 NEXTI
300 PRINTINT(TI-T1)/60
400 FORI=ØTON-1
410 IFA(I)>A(I+1)THENPRINTI,A(I),A(I+1)
420 NEXT
430 PRINT"ALLES AUFSTEIGEND SORTIERT!"
```

Dieses BASIC-Programm sortiert Zahlenfelder. Man kann es leicht auf das Sortieren von Strings umschreiben. Der Algorithmus steckt in den Zeilen 120 bis 180. Das Verfahren ist von der Ordnung $O(n^2)$, bei Verdoppelung der Anzahl der zu sortierenden Strings vervierfacht sich die Rechenzeit. Es ist im Vergleich zum bekannten Bubble-Sort etwa 3mal so schnell, da erheblich weniger Vertauschungen vorgenommen werden müssen. Es ist immer noch schneller als schnell gemachte Versionen des Bubble-Sorts und es ist in der Programmierung nicht aufwenidiger als die Reinversion des Bubble-Sort.

Folgende Einschränkungen werden für die Assemblerversion dieses Programms getroffen:

Sortiert wird ein Feld A$(.).
Es muß ein erstes Feld dimensioniert worden sein.
Es muß eindimensional sein.

Dieses wird vom Programm überprüft, und bei Nichterfüllung beendet das Programm automatisch seine Arbeit.

Eine Anmerkung:
Auf Strings kann man zugreifen, wenn man deren Startadresse kennt. Für jeden String gibt es an der entsprechenden Stelle des Feldes A $(.) einen Pointer auf die Länge des Strings und einen Pointer auf den Beginn des Stings. Über die Zeropage kann man damit über zweifaches indirektes Zugreifen byteweise einen Strring holen. Um den Platz für die Speicherung der Pointer in der Zeropage zu erhalten, wurde ein Teil der Pointer des BASIC-Interpreters ausgelagert und nach Durchlauf des Programmes wiederhergestellt. Dies verursacht keine Probleme, da diese Werte in der Assemblerroutine nicht benötigt und vor dem Rücksprung ins BASIC wiederhergestellt werden.

Das Programm:

```
027A  A0 00     LDY #00
027C  B1 2C     LDA (2C),Y   TEST AUF FELD A$(.)
027E  C9 41     CMP #41
0280  F0 03     BEQ 0285
```

```
0282  4C 96 03   JMP 0396
0285  C8         INY
0286  B1 2C      LDA (2C),Y
0288  C9 80      CMP #80
028A  F0 03      BEQ 028F
028C  4C 96 03   JMP 0396
028F  A0 04      LDY #04
0291  B1 2C      LDA (2C),Y
0293  C9 01      CMP #01      TEST AUF ANZAHL DIMENSIONEN
0295  F0 03      BEQ 029A
0297  4C 96 03   JMP 0396
029A  A0 02      LDY #02
029C  18         CLC
029D  A5 2C      LDA 2C       FELDENDE BERECHNEN
029F  71 2C      ADC (2C),Y
02A1  8D 87 03   STA 0397     UND SPEICHERN IN $0397
02A4  A5 2D      LDA 2D
02A6  C8         INY
02A7  71 2C      ADC (2C),Y
029A  8D 98 03   STA 0398     HOHERWERTIGES BYTE
02AC  38         SEC
02AD  AD 97 03   LDA 0397     VON BEGINN DES NACHSTEN FELDES
02B0  E9 03      SBC #03      UMRECHNEN AUF
02B2  8D 97 03   STA 0397     ENDE DES FELDES A$(.)
02B5  AD 98 03   LDA 0398
02B8  E9 00      SBC #00
02BA  8D 98 03   STA 0398

02BD  A2 03      LDX #03      PLATZ IN ZERO PAGE SCHAFFEN
02BF  B5 32      LDA 32,X
02C1  9D 08 02   STA 0208,X
02C4  CA         DEX
02C5  10 F8      BPL 02BF     TRANSPORTSCHLEIFE
02C7  18         CLC
02C8  A5 2C      LDA 2C
02CA  69 07      ADC #07      VARIABLE I SPEICHERN
02CC  85 01      STA 01       IN $0001 FF.
02CE  A5 2D      LDA 2D
02D0  69 00      ADC #00
02D2  85 02      STA 02
02D4  A5 01      LDA 01       VARIABLE K SPEICHERN
02D6  85 32      STA 32       NACH $0032 FF.
02D8  A5 02      LDA 02
02DA  85 32      STA 33
02DC  18         CLC
02DD  A5 32      LDA 32
02DF  69 03      ADC #03      VARIABLE J SPEICHERN
02E1  85 34      STA 34       NACH $0034 FF.
02E3  A5 33      LDA 33
02E5  69 00      ADC #00
02E7  85 35      STA 35
02E9  A0 02      LDY #02
02EB  B1 32      LDA (32),Y   STARTADRESSE VON A$(K)
02ED  85 57      STA 57       NACH $0056 FF.
02EF  88         DEY
02F0  B1 32      LDA (32),Y
02F2  85 56      STA 56
02F4  A0 02      LDY #02
02F6  B1 34      LDA (34),Y   STARTADRESSE VON A$(J)
```

```
02F8  85 59      STA 59       NACH $0058 FF.
02FA  88         DEY
02FB  B1 34      LDA (34),Y
02FD  85 58      STA 58
02FF  DEY
0300  B1 32      LDA (32),Y   LÄNGE VON A$(K)=Ø ?
0302  C9 00      CMP #00
0304  F0 56      BEQ 035C
0306  8D 99 03   STA 0399     LÄNGE SPEICHERN
0309  D1 34      CMP (34),Y   VERGLEICH MIT LÄNGE A$(J)
030B  30 09      BMI 0316
030D  B1 34      LDA (34),Y
030F  C9 00      CMP #00      LÄNGE A$(J) =Ø ?
0311  F0 1D      BEQ 0330     LÄNGE  SPEICHERN, WENN LÄNGE
0313  8D 99 03   STA 0399     A$(J) KLEINER LÄNGE A$(K)
0316  B1 58      LDA (58),Y   BYTEWEISER VERGLEICH DER
0318  D1 56      CMP (56),Y   ZEICHENFOLGEN
031A  90 14      BCC 0330     IST KLEINER
031C  F0 03      BEQ 0321     IST GLEICH
031E  4C 43 03   JMP 0343     IST GRÖSSER
0321  C8         INY
0322  CC 99 03   CPY 0399
0325  90 EF      BCC 0316     WEITER VERGLEICHEN
0327  A0 00      LDY #00
0329  B1 32      LDA (32),Y   BEI ÜBEREINSTIMMUNG PRÜFEN,
032B  CD 99 03   CMP 0399     WELCHES DER KÜRZERE STRING IST
032E  F0 13      BEQ 0343
0330  A5 34      LDA 34       ANWEISUNG: K=J
0332  85 32      STA 32       ALSO UMSPEICHERN
0334  A5 35      LDA 35
0336  85 33      STA 33
0338  A0 01      LDY #01
033A  B1 32      LDA (32),Y   STARTADRESSE UMSPEICHERN
033C  85 56      STA 56
033E  C8         INY
033F  B1 32      LDA (32),Y
0341  85 57      STA 57
0343  18         CLC
0344  A5 34      LDA 34       J ERHÖHEN
0346  69 03      ADC #03
0348  85 34      STA 34
034A  A5 35      LDA 35
034C  69 00      ADC #00
034E  85 35      STA 35
0350  CD 98 03   CMP 0398     PRÜFEN AUF FELDENDE
0353  90 9F      BCC 02F4
0355  AD 97 03   LDA 0397
0358  C5 34      CMP 34
035A  B0 98      BCS 02F4
035C  A0 02      LDY #02      TAUSCH DER POINTER,
035E  B1 01      LDA (01),Y   DAMIT UMSPEICHERN VON
0360  AA         TAX          A$(I) UND A$(K)
0361  B1 32      LDA (32),Y
0363  91 01      STA (01),Y
0365  8A         TXA
0366  91 32      STA (32),Y
0368  88         DEY
```

```
0369  10 F3      BPL 035E
036B  18         CLC
036C  A5 01      LDA 01        I ERHÖHEN
036E  69 03      ADC #03
0370  85 01      STA 01
0372  A5 02      LDA 02
0374  69 00      ADC #00
0376  85 02      STA 02
0378  CD 98 03   CMP 0398      PRÜFEN AUF FELDENDE
037B  B0 03      BCS 0380              V
037D  4C D4 02   JMP 02D4
0380  A5 01      LDA 01
0382  CD 97 03   CMP 0397
0385  F0 05      BEQ 038C
0387  B0 03      BCS 038C
0389  4C D4 02   JMP 02D4
038C  A2 03      LDX #03
038E  BD 08 02   LDA 0208,X   ALTE WERTE DER ZERO PAGE
0391  95 32      STA 32,X     WIEDERHERSTELLEN
0393  CA         DEX
0394  10 F8      BPL 038E
0396  60 RTS
```

Einige Ergebnisse

Mit 128 Elementen und einer mittleren Länge von 4 Bytes benötigte

das BASIC-Programm	ca. 85 Sekunden
die Assemblerroutine	ca. 1,3 Sekunden.

Mit 512 Elementen und einer mittleren Länge von 10 Bytes benötigte

das BASIC-Programm	ca. 23 Minuten
die Assembler-Routine	ca. 22 Sekunden.

Bei der Anwendung des Programms muß man beachten, daß leere Strings an den Anfang des Feldes wandern. Bei einem Feld mit 100 Elementen, davon nur 70 mit Strings besetzt, tauchen nach der Sortierung erst 30 leere Strings und danach die sortierten Strings auf. Strings der Länge 0 sind natürlich alphabetisch kleiner als Strings der Länge 1! Das Assemblerprogramm prüft auf Länge 0. Es prüft außerdem, daß bei unterschiedlicher Länge aber Übereinstimmung im vorderen Teil der Strings (Beispiel: ABA in Vergleich zu AB) der kürzere String der alphabetisch kleinere ist.

Zahlen kann man über diese Assembler-Routine auch sortieren, man muß sie vorher in Strings umwandeln. Dabei tritt folgender Effekt auf: Die Zahlen von 1 bis 100 werden in folgende Reihenfolge sortiert

1, 10, 100, 11, 12 ... 19, 2, 20 usw..

Sie sind damit korrekt sortiert, wie man an der Schreibweise erkennt. Diesen Effekt benötigt man bei Sortierung von Kontonummern in einem Buchhaltungsprogramm oder bei Sortierung von Anschriften nach Postleitzahlen.

Will man Anschriften nach Postleitzahlen sortieren, so liest man die Postleitzahlen von der Diskette in das Feld A$(.) ein und hängt an jeden String folgendes an:

```
A$(I) = PZ$ + CHR$(Ø) + CHR$(SP) + CHR$(SE)
```

Die letzten 3 Bytes bedeuten

1) Trennungszeichen
2) Spur-Nummer des zugehörigen Satzes
3) Sektor-Nummer des Satzes.

Nach der Sortierung kann man dann ebenfalls auf den entsprechenden Satz zugreifen.

Möglich ist ebenfalls folgender Weg:

```
A$(I) = PZ$ + CHR$(Ø) + STR$(LAUFINDEX)
```

Über den Laufindex kann man dann ebenfalls auf den entsprechenden Satz zugreifen.

Falls nach mehreren Kriterien sortiert werden soll, muß man die Elemente zu einem String sammenbauen.

```
A$(I) = PZ$ + CHR$(Ø) + ...
```

Falls nach mehr als 2 Elementen sortiert werden soll, müssen die einzelnen Teile allerdings die gleiche Länge haben, also 2. Element mit fester Länge, 3. Element mit fester Länge usw.. Eine feste Länge benötigt man auch, wenn Zahlen numerisch sortiert werden sollen. Dazu muß man die Strings von links mit Blanks auffüllen und danach sortieren.

Das Programm sortiert nur aufsteigend. Will man absteigend sortieren, muß man nach dem Sortiervorgang nur in der umgekehrten Folge auf die Elemente des Feldes zugreifen, also

```
FOR I = N TO 0 STEP -1.
```

Das Programm sortiert allerdings nicht nach dem 1. Kriterium aufsteigend und nach dem 2. Kriterium absteigend. Dies kann man nur durch mehrere Sortiervorgänge erreichen. Dazu wird erst nach dem 1. Kriterium sortiert. Das Ergebnis wird abgespeichert. Alle Elemente mit gleichen Schlüsseln werden einem neuen Sortiervorgang, jetzt mit dem 2. Kriterium unterworfen usw.. Dies ist auch eine Alternative zu der Sortierung nach mehreren Kriterien in einem Durchgang (s.o.).

Das Programm benötigt nur die beiden Kassettenpuffer. Es beansprucht keine Bytes aus dem Bereich des BASIC oder der Variablen. Es arbeitet mit genügender Geschwindigkeit, da der Aufbau des Feldes A$(.) mehr Zeit beansprucht als der eigentliche Sortiervorgang.

Literatur

Barrodale, Roberts, Ehle: Einführung in die Anwendung von Digitalrechnern, München 1973.
Aho, Hopcroft, Ullman: The Design and Analysis of Computer Algorithms, London 1975.
Wirth: Algorithmen und Datenstrukturen, Stuttgart 1979. ¤

PET 2001-Platine vollbestückt, einwandfreier Zustand zu verkaufen, ferner CBM ROM-Satz (28 Pin). Peter Arps, Brockdorffstraße 5, 2000 Hamburg 73, Tel.: 040-672 05 60.

Wer kann Schaltplan - auch teilweise - von CBM 3032 liefern? Klaus Schuenemann, Tel. 0221-23 60 47 (d) bzw. 760 19 31 (p).

AIM-Entwicklungssystem, 32 KBytes, HF-Monitorausgang, 64 x 16 Zeichen; BASIC + Erw.-EPROM, Assembler 8 KBytes, PASCAL-Compiler, Netzteil plus/minus 5 V, plus/minus 12 V, 24 V mit Lüfter, QWERTZ-Tastatur, alles im AIM-Kunststoffgehäuse, viel Soft- u. Hardware, NP 3500 DM. VB 2400 DM. Martin Roßmüller, Tel.: 0228 - 22 48 37.

Computhink Expandamem 32 K. Beschaltungsvorschlag für Betrieb am AIM 65 gesucht. Roland Löhr, Tel.: 04 102 - 55 816. ¤

Zum bevorstehenden Jahreswechsel
wünscht der Herausgeber allen Lesern persönlich und beruflich alles Gute, verbunden mit dem Dank für die Zusammenarbeit im abgelaufenen Jahr.

Auf ein friedliches und glückliches 1981!

Christian Streicher, 8034 Germering

Assembler Cross Reference Map - AIM 65

Das vorliegende Programm ermöglicht die Ausgabe einer echten Symbol-Cross-Reference Map. Die zu suchenden Symbole werden der Symboltabelle des Assemblers entnommen. Der eigentliche Suchvorgang erfolgt in zwei Durchläufen. Im ersten Durchlauf wird der Arbeitsspeicherbereich, in dem sich der Quelltext befinden muß, nach der Definitionszeile des Symbols abgesucht. Ist das Symbol gefunden worden, so beginnt das Programm den Quelltext zeilenweise nach den aufrufenden Textstellen abzusuchen. Symbole, die nach einem ';' erscheinen, werden nicht zur Auslistung gebracht. Es werden jedoch alle Symbole erkannt, die nicht in Spalte 0 des Quelltextes beginnen, auch die Symbole, die in arithmetischen Verknüpfungen aufgerufen werden. Ein in Spalte 0 beginnendes Symbol wird unter 'DEFINED' zur Auslistung gebracht.

Die Symbole werden unmittelbar nach dem Auffinden an eine frei definierbare List-Routine geleitet (OUT1), so daß keinerlei Speicherplatz reserviert werden muß. Die vollständige Auslistung einer Symbol-Reference aus einem 20 KByte Quelltext benötigt allerdings ca. 10 bis 20 Sekunden. - Die Zeilennummernzuordnung erfolgt nach dem 6502-Standard. Das bedeutet, daß in einer Zeile, in der lediglich ein (SPACE) - (CARRIAGE RETURN) steht, keine Zeilennummer zugeordnet wird.

Aufruf der Routine
Die Routine darf erst nach Beendigung des Assembler-Laufes aufgerufen werden, da erst zu diesem Zeitpunkt sichergestellt ist, daß die Symboltabelle angelegt und zur Abfrage frei ist. Der Einsprung in diese Routine kann im einfachsten Fall über eine Benutzerfunktionstaste (z.B. F1) erfolgen. Eine elegantere Lösung wurde vom Autor durch Hinzufügen von Sonderoptionen (vgl. SIEMENS PC 100 Assembler-Handbuch) realisiert. Die Reformatierung des Listings wird über die Option 'CNT' (Count) gesteuert. Nach dem Erkennen der .END-Anweisung fragt das Steuerprogramm ein zuvor in der Zero Page gesetztes Flag ab und springt, entsprechend dem Wert dieses Flags in eine Routine, die das Ordnen der Symboltabelle besorgt. Ist zuvor im Quelltext die Anweisung .OPT COU (Cross Reference Output) gegeben worden, so wird die Cross Reference Map erstellt.

Um die Ausgabebreite dieser Cross Reference Map möglichst variabel zu gestalten, muß vor dem Aufruf der Routine ein Flag in Zelle F8 gesetzt werden. Ist der Wert dieses Flags gleich 1, so werden 60 Character ausgegeben (z.B. für Videoausgabe), andernfalls werden 79 Character zur Ausgabe geleitet, z.B. für Druckerausgaben.

Um das Lesen der Cross Refernce Map zu erleichtern, wird nach je 73 Zeilen eine neue Zeile begonnen. Zu Beginn jeder neuen Seite werden die Überschriftzeilen mitausgedruckt. Die Zahl der Zeilen pro Seite ist für den CBM-3023 Drucker programmiert, sie kann jedoch durch Umschreiben der Zeilen 101 und 226 beliebig geändert werden.

```
PAGE 01
LINE # LOC  CODE              LINE

0001  0000                     .OPT CNT
0002  0000                   ;**********************************
0003  0000                   ;**********************************
0004  0000                   ;***  CROSS-REFERENCE-MAP       ***
0005  0000                   ;***  --------------------      ***
0006  0000                   ;***                            ***
0007  0000                   ;*** COPYRIGHT BY :             ***
0008  0000                   ;***       CHRISTIAN STREICHER  ***
0009  0000                   ;*** OCT./25/1980  REV.:1.1     ***
```

```
0012  0000                    .OPT SYM
0013  0000                    .OPT COU


0014  0000              ;*** VARIABLES ***

0015  0000              OUTFLG =$F8
0016  0000              ; 1=GENERATES 60 COLUMNS OUTPUT (FOR TV)
0017  0000              ; 2=GENERATES 79 COLUMNS OUTPUT (FOR PRINTER)
0018  0000              BUFF   =$F2
0019  0000              NOSYM  =$0B
0020  0000              STSAVE =$3A
0021  0000              NOWLN  =$DF
0022  0000              ETEXT  =$E3
0023  0000              STORE1 =$D0
0024  0000              PAGE   =$FB
0025  0000              CR     =$0D
0026  0000              ISYM   =$2A
0027  0000              CNTR1  =$FD
0028  0000              SPC    =$20

0029  0000              OUT1   *=*             ; VECTOR TO USER OUTPUT
0030  0000              MREAD  =$FAD0
0031  0000              RCHEK  =$E907                        ROUTINE

0032  0000              ;*** PROGRAM START ***
0033  0000                    *=$9800


0034  9800  A5 0B       CROSS  LDA NOSYM       ; SAVE SYM.-TABLE POINTER
0035  9802  85 D0              STA STORE1
0036  9804  A5 0C              LDA NOSYM+1
0037  9806  85 D1              STA STORE1+1
0038  9808  A5 3A              LDA STSAVE
0039  980A  85 D2              STA STORE1+2
0040  980C  A5 3B              LDA STSAVE+1
0041  980E  85 D3              STA STORE1+3
0042  9810  20 16 99           JSR NEWCR
0043  9813  20 5D 99    NXTSYM JSR CLEAR
0044  9816  A5 0C              LDA NOSYM+1
0045  9818  38                 SEC
0046  9819  E9 01              SBC #1
0047  981B  85 0C              STA NOSYM+1
0048  981D  A5 0B              LDA NOSYM
0049  981F  E9 00              SBC #0
0050  9821  85 0B              STA NOSYM
0051  9823  10 14              BPL GOON
0052  9825  20 BC 99           JSR SYDO
0053  9828  A5 D0              LDA STORE1
0054  982A  85 0B              STA NOSYM       ; RESTORE SYM.-TABLE REG.
0055  982C  A5 D1              LDA STORE1+1
0056  982E  85 0C              STA NOSYM+1
0057  9830  A5 D2              LDA STORE1+2

0058  9832  85 3A              STA STSAVE
0059  9834  A5 D3              LDA STORE1+3
0060  9836  85 3B              STA STSAVE+1
0061  9838  60                 RTS
```

```
PAGE 02
LINE # LOC  CODE            LINE

0062  9839  A5 FC      GOON   LDA PAGE+1      ; CHECK IF NEW PAGE IS
0063  983B  C9 46             CMP #70                  NESSESARY
0064  983D  30 03             BMI *+5
0065  983F  20 16 99          JSR NEWCR
0066  9842  A9 0D             LDA #CR
0067  9844  20 00 00          JSR OUT1
0068  9847  E6 FC             INC PAGE+1
0069  9849  A0 00             LDY #0
0070  984B  B1 3A      LOOP2  LDA (STSAVE),Y ; COPY SYM INTO BUFFER
0071  984D  99 2A 00          STA ISYM,Y
0072  9850  20 00 00          JSR OUT1
0073  9853  C8               INY
0074  9854  C0 06             CPY #6
0075  9856  D0 F3             BNE LOOP2
0076  9858  20 EA 99          JSR SPC2        ; PRINT TWO SPC
0077  985B  20 84 99    SEA0  JSR SEADEF      ; CHECK THE LINE FOR STRING
0078  985E  B0 0D             BCS FND1        ; FOUND
0079  9860  20 D0 FA    SEA2  JSR MREAD       ; SEARCH FOR NEXT (CR)
0080  9863  C9 0D             CMP #CR
0081  9865  D0 F9             BNE SEA2
0082  9867  20 E6 98          JSR CHKCR
0083  986A  4C 5B 98          JMP SEA0

0084  986D  20 D9 99    FND1  JSR OUTLIN      ; PRINT LINE NUMBER
0085  9870  20 00 00          JSR OUT1        ; PRINT 3 SPC
0086  9873  20 5D 99          JSR CLEAR       ; CLEAR BUFFER & POINTER
0087  9876  A9 0F             LDA #15         ; SET ROW COUNTER
0088  9878  85 FD             STA CNTR1

0089  987A  A5 F8      LOOP4  LDA OUTFLG
0090  987C  29 01             AND #%00000001
0091  987E  F0 04             BEQ CR1
0092  9880  A9 3C             LDA #60
0093  9882  D0 02             BNE CR2
0094  9884  A9 4A      CR1    LDA #74         ; CHECK IF NEW LINE REQUIRED
0095  9886  C5 FD      CR2    CMP CNTR1
0096  9888  10 1C             BPL CR3
0097  988A  A9 0D             LDA #CR
0098  988C  20 00 00          JSR OUT1
0099  988F  E6 FC             INC PAGE+1      ; UPDATE LINE COUNTER
0100  9891  A5 FC             LDA PAGE+1      ; AND CHECK FOR NEW PAGE
0101  9893  C9 46             CMP #70
0102  9895  30 03             BMI CR4
0103  9897  20 16 99          JSR NEWCR
0104  989A  A9 20      CR4    LDA #SPC
0105  989C  A2 0F             LDX #15
0106  989E  86 FD             STX CNTR1       ; SET ROW COUNTER
0107  98A0  20 00 00          JSR OUT1
0108  98A3  CA               DEX
0109  98A4  D0 FA             BNE *-4
0110  98A6  20 D0 FA    CR3    JSR MREAD       ; GET THE REFERENCE
0111  98A9  20 E6 98          JSR CHKCR
0112  98AC  C9 00             CMP #0
0113  98AE  F0 21             BEQ M08
0114  98B0  C9 3B             CMP #';
```

```
PAGE 03
LINE # LOC  CODE             LINE

0115  98B2  F0 23            BEQ CR7
0116  98B4  C9 3C            CMP #'<
0117  98B6  F0 10            BEQ M09
0118  98B8  C9 3D            CMP #'=
0119  98BA  F0 0C            BEQ M09
0120  98BC  C9 3E            CMP #'>
0121  98BE  F0 08            BEQ M09
0122  98C0  C9 30            CMP #'0         ; SYMBOLS NEVER BEGIN WITH
0123  98C2  B0 E2            BCS CR3         ; NON ALPA CHR.
0124  98C4  C9 0D            CMP #CR
0125  98C6  F0 DE            BEQ CR3
0126  98C8  20 84 99   M09    JSR SEADEF
0127  98CB  B0 3C            BCS M13         ; FOUND A MATCH
0128  98CD  C9 00            CMP #0          ; END OF BUFFER
0129  98CF  D0 D5            BNE CR3
0130  98D1  20 76 99   M08    JSR CLEAR1      ; CLEAR POINTER
0131  98D4  4C 13 98          JMP NXTSYM      ; CHECK NEXT SYMBOL

0132  98D7  20 D0 FA   CR7    JSR MREAD       ; GET NEXT LINE
0133  98DA  F0 F5            BEQ M08         ; END OF TEXT BUFFER
0134  98DC  C9 0D            CMP #CR
0135  98DE  D0 F7            BNE CR7
0136  98E0  20 E6 98          JSR CHKCR
0137  98E3  4C A6 98          JMP CR3

0138  98E6                  ;*****  CHECK IF BLANK CARD IS IN BUFFER *****
0139  98E6                  ;*****  IF ONE THEN EXECUTE GARBAGE COLLECTOR ***

0140  98E6  48         CHKCR  PHA             ; SAVE SYMBOL
0141  98E7  C9 0D            CMP #CR         ; CHECK FIRST (CR)
0142  98E9  D0 11            BNE RT1
0143  98EB  A0 00            LDY #0
0144  98ED  A9 20            LDA #SPC        ; CHECK (SPC)
0145  98EF  D1 DF            CMP (NOWLN),Y
0146  98F1  D0 09            BNE RT1
0147  98F3  A9 0D            LDA #CR
0148  98F5  C8               INY
0149  98F6  D1 DF            CMP (NOWLN),Y   ; CHECK SECOND (CR)
0150  98F8  D0 02            BNE RT1
0151  98FA  68               PLA
0152  98FB  60               RTS
0153  98FC  68         RT1    PLA             ; CHECK IF LINE NUMBER MUS1
0154  98FD  48               PHA             ; BE ADDED
0155  98FE  C9 0D            CMP #CR
0156  9900  D0 05            BNE RT2
0157  9902  A2 01            LDX #1
0158  9904  20 C3 99          JSR COMP
0159  9907  68         RT2    PLA
0160  9908  60               RTS

0161  9909  20 D9 99   M13    JSR OUTLIN      ; PRINT LINE NUMBER
0162  990C  A5 FD            LDA CNTR1       ; UPDATE ROW COUNTER
0163  990E  18               CLC
0164  990F  69 06            ADC #6
0165  9911  85 FD            STA CNTR1
0166  9913  4C 7A 98          JMP LOOP4
```

```
167  9916  20 AC 99   NEWCR  JSR COUDON     ; PRINT NEW PAGE
168  9919  A2 05             LDX #5
169  991B  86 FC             STX PAGE+1
170  991D  A2 33             LDX #51         ; PRINT HEADLINES
171  991F  BD 29 99  LOOP1   LDA TEXT4,X
172  9922  20 00 00          JSR OUT1
173  9925  CA                DEX
174  9926  10 F7             BPL LOOP1
175  9928  60                RTS

176  9929  0D         TEXT4  .BYT CR,CR,'SECNEREFER  '
177  9937  44 45 4E           .BYT 'DENIFED',' LOBMYS'
178  9945  0D                .BYT CR,CR,'ECNEREFER '
179  9951  53 53 4F           .BYT 'SSORC LOBMYS'
180  995D  A2 05       CLEAR  LDX #5          ; CLEAR BUFFER AND
181  995F  A9 00              LDA #0          ; REINITIALISE POINTER FOR NEXT
182  9961  95 F2       LOOP3  STA BUFF,X      ; SYMBOL
183  9963  CA                 DEX
184  9964  10 FB              BPL LOOP3
185  9966  85 FD              STA CNTR1
186  9968  A2 01              LDX #1
187  996A  20 C3 99           JSR COMP
188  996D  A5 E3              LDA ETEXT
189  996F  85 DF              STA NOWLN
190  9971  A5 E4              LDA ETEXT+1
191  9973  85 E0              STA NOWLN+1
192  9975  60                 RTS

193  9976  A5 3A       CLEAR1 LDA STSAVE      ; UPDATE NUMBER OF SYMBOLS
194  9978  18                CLC
195  9979  69 08              ADC #8
196  997B  85 3A              STA STSAVE
197  997D  A5 3B              LDA STSAVE+1
198  997F  69 00              ADC #0
199  9981  85 3B              STA STSAVE+1
200  9983  60                 RTS

201  9984  A0 00       SEADEF LDY #0          ; SEARCH IF VALID STRING OCCURS
202  9986  F0 07              BEQ G224        ; CURRENT LINE          IN
203  9988  B9 2A 00    G210   LDA ISYM,Y      ; CHECK IF CHR. IS OK
204  998B  C9 20              CMP #SPC
205  998D  F0 10              BEQ G230
206  998F  B1 DF       G224   LDA (NOWLN),Y
207  9991  C9 3B              CMP #';         ; ALL DONE WITH COMMENT CARD
208  9993  F0 15              BEQ DONE
209  9995  D9 2A 00           CMP ISYM,Y      ; CHECK CHARACTER
210  9998  D0 10              BNE DONE
211  999A  C8                 INY
212  999B  C0 06              CPY #6
213  999D  D0 E9              BNE G210
214  999F  B1 DF       G230   LDA (NOWLN),Y
215  99A1  C9 30              CMP #'0
216  99A3  B0 05              BCS DONE
217  99A5  20 07 E9           JSR RCHEK
218  99A8  38                 SEC             ; STRING FOUND SO SET C=1
219  99A9  60                 RTS

220  99AA  18           DONE  CLC             ; NONE FOUND SO SET C=0
221  99AB  60                 RTS
```

```
0222  99AC  A9 0D      COUDON LDA #CR         ; BEGIN NEW PAGE WITHOUT...
0223  99AE  A6 FC             LDX PAGE+1      ; HEADLINE
0224  99B0  20 00 00          JSR OUT1
0225  99B3  E8               INX
0226  99B4  E0 49             CPX #73
0227  99B6  30 F8             BMI *-6
0228  99B8  A2 00             LDX #0          ; CLEAR POINTER FOR SYMBOL
0229  99BA  86 FC             STX PAGE+1
0230  99BC  A5 37      SYDO   LDA $37         ; CLEAR LIST-OUT OPTION
0231  99BE  09 10             ORA #16
0232  99C0  85 37             STA $37
0233  99C2  60               RTS

0234  99C3            ;*** SUBROUTINE GENERATES NEXT LINE NUMBER ***

0235  99C3  B5 F2      COMP   LDA BUFF,X
0236  99C5  18               CLC
0237  99C6  69 01             ADC #1
0238  99C8  C9 0A             CMP #$0A        ; CHECK IF NUMBER EXCEEDS 10
0239  99CA  90 0A             BCC LOW
0240  99CC  A9 00             LDA #0          ; CLEAR DIGIT
0241  99CE  20 D6 99          JSR LOW
0242  99D1  E8               INX
0243  99D2  20 C3 99          JSR COMP        ; CHECK NEXT DECADE
0244  99D5  60               RTS
0245  99D6  95 F2      LOW    STA BUFF,X
0246  99D8  60               RTS

0247  99D9            ;*** PRINT LINE NUMBER ***

0248  99D9  A2 04      OUTLIN LDX #4
0249  99DB  B5 F2      CR5    LDA BUFF,X
0250  99DD  18               CLC             ; CONVERT NUMBER TO ASCII
0251  99DE  69 30             ADC #$30
0252  99E0  20 00 00          JSR OUT1
0253  99E3  CA               DEX
0254  99E4  D0 F5             BNE CR5
0255  99E6  20 EA 99          JSR SPC2        ; PRINT TWO SPACE
0256  99E9  60               RTS

0257  99EA            ;*** SUBROUTINE GENERATES TWO SPACE ***

0258  99EA  A9 20      SPC2   LDA #SPC
0259  99EC  20 00 00          JSR OUT1
0260  99EF  4C 00 00          JMP OUT1
0261  99F2                    .END ASSMI
```

SYMBOL TABLE

SYMBOL	VALUE								
BUFF	00F2	CHKCR	98E6	CLEAR	995D	CLEAR1	9976	CNTR1	00FD
COMP	99C3	COUDON	99AC	CR	000D	CR1	9884	CR2	9886
CR3	98A6	CR4	989A	CR5	99DB	CR7	98D7	CROSS	9800
DONE	99AA	ETEXT	00E3	FND1	986D	G210	9988	G224	998F
G230	999F	GOON	9839	ISVM	002A	LOOP1	991F	LOOP2	984B
LOOP3	9961	LOOP4	987A	LOW	99D6	M08	98D1	M09	98C8
M13	9909	MREAD	FAD0	NEWCR	9916	NOSVM	000B	NOWLN	00DF

```
NXTSYM  9813   OUT1    0000   OUTFLG  00F8   OUTLIN  99D9   PAGE    00FB
RCHEK   E907   RT1     98FC   RT2     9907   SEA0    985B   SEA2    9860
SEADEF  9984   SPC     0020   SPC2    99EA   STORE1  00D0   STSAVE  003A
SVDO    99BC   TEXT4   9929
     .END ASSMI
ERRORS= 0000

SYMBOL CROSS REFERENCE

SYMBOL DEFINED  REFERENCES


BUFF    0018   0182  0235  0245  0249
CHKCR   0140   0082  0111  0136
CLEAR   0180   0043  0086
CLEAR1  0193   0130
CNTR1   0027   0088  0095  0106  0162  0165  0185
COMP    0235   0158  0187  0243
COUDON  0222   0167
CR      0025   0066  0080  0097  0124  0134  0141  0147  0155  0176  0176
               0178  0178  0222
CR1     0094   0091
CR2     0095   0093
CR3     0110   0096  0123  0125  0129  0137
CR4     0104   0102
CR5     0249   0254
CR7     0132   0115  0135
CROSS   0034                        ISYM    0026   0071  0203  0209
DONE    0220   0208  0210  0216     LOOP1   0171   0174
ETEXT   0022   0188  0190           LOOP2   0070   0075
FND1    0084   0078                LOOP3   0182   0184
G210    0203   0213                LOOP4   0089   0166
G224    0206   0202                LOW     0245   0239  0241
G230    0214   0205                M08     0130   0113  0133
GOON    0062   0051                M09     0126   0117  0119  0121

M13     0161   0127
MREAD   0030   0079  0110  0132
NEWCR   0167   0042  0065  0103
NOSYM   0019   0034  0036  0044  0047  0048  0050  0054  0056
NOWLN   0021   0145  0149  0189  0191  0206  0214
NXTSYM  0043   0131
OUT1    0029   0067  0072  0085  0098  0107  0172  0224  0252  0259  0260

OUTFLG  0015   0089
OUTLIN  0248   0084  0161
PAGE    0024   0062  0068  0099  0100  0169  0223  0229
RCHEK   0031   0217
RT1     0153   0142  0146  0150
RT2     0159   0156
SEA0    0077   0083
SEA2    0079   0081
SEADEF  0201   0077  0126
SPC     0028   0104  0144  0204  0258
SPC2    0258   0076  0255
STORE1  0023   0035  0037  0039  004₁  0053  0055  0057  0059
STSAVE  0020   0038  0040  0058  0060  0070  0193  0196  0197  0199
SVDO    0230   0052
TEXT4   0176   0171
```

Steuerung - elegant per Assembler

Meß- und Steuerungsaufgaben verlangen ein bitweises Denken. Man muß sich stets vor Augen halten, welche Aufgaben den Pins der benutzten Interfacebausteine zugewiesen sind.

Im folgenden Aufsatz werden einige Anregungen gegeben, wie man die Verarbeitungsfähigkeit der 6502-Assembler für erklärte Symbole und für die Berechnung von 'Ausdrücken' geschickt benutzen kann, um mehr in Bedingungen als in Bits programmieren zu können. Die Ausführungen sind zugleich eine Anwendung der logischen Maschinenbefehle AND, ORA und EOR sowie der Assembler-Anweisung = (EQUATE). Die Bildung von Ausdrücken wird beschrieben.

Prinzipien der Interfaceprogrammierung

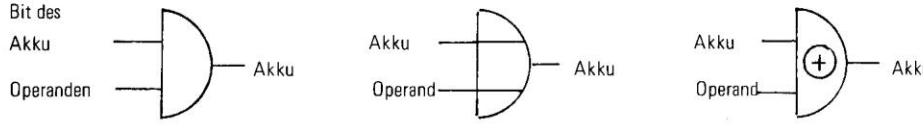
Die Programmierung von Steuerungen ist ein Teilabschnitt der allgemeinen Interfaceprogrammierung. Die nachfolgenden Beispiele betreffen die 6522 VIA, weil dieser Interfacebaustein eine große Verbreitung hat und weil er die typischen Familienmerkmale hat (siehe auch Heft 7: Die VIA 6522): Bei den 65er und 68er Mikroprozessoren gibt es keine besonderen Ein- und Ausgabebefehle. Interfacebausteine mit ihren Registern werden vielmehr als ganz normale Speicheradressen angesprochen und mit LDA-Befehlen gelesen und mit STA-Befehlen beschrieben.

Weiterhin ist typisch: Ein Interfacebaustein trägt im allgemeinen 2 Ports (E/A-Schnittstellen) mit je 8 Pin. Dabei kann jedes Pin individuell als Eingang oder Ausgang geschaltet werden. Die Einrichtung dieser Funktion erfolgt durch initialisierendes Einschreiben eines Bitmusters in das Datenrichtungsregister. Alle mit '1' beschriebenen Bitpositionen dienen hernach als Ausgang, alle Bitpositionen mit '0' als Eingangspins. Der eigentliche Datenaustausch findet an den Ports (mit eigener Adresse) durch LDA- und STA-Befehle statt.

Ein Port ist also ein 8 Bit breites E/A-Register. Bei Steuerungen möchte man nun im allgemeinen nicht 8 Pins zugleich umschalten, sondern nur eine Untermenge davon, die dem gegenwärtigen Prozeßzustand entspricht. Hierbei kommen die logischen Befehle vorteilhaft zum Einsatz.

Die logischen Befehle

Zunächst zu den logischen Befehlen: Ihre Funktion ist im Programmierhandbuch beschrieben. Im Gegensatz zu fast allen anderen Befehlen wirken sie bitweise, Akkumulator gegen Inhalt einer Speicherzelle oder gegen Direktoperand. Das Ergebnis der bitweisen Verknüpfung steht nach der Operation im Akkumulator. In den Symbolen der Digitaltechnik muß man sich 8 parallele Gatter wie folgt denken:



UND ODER Exkl. ODER

Für das bitweise Ergebnis gelten folgende Wahrheitstabellen:

Operand \ Akku-Bit	0	1
0	0	0
1	0	1

UND

Operand \ Akku-Bit	0	1
0	0	1
1	1	1

ODER

Operand \ Akku-Bit	0	1
0	0	1
1	1	0

Exkl. ODER

In verbaler Erklärung: Das Bit-Ergebnis einer ODER-Verknüfung ist dann gleich '1', wenn einer der Eingänge oder wenn beide gleich '1' sind. Bei der UND-Verknüfung ist das gebnis nur dann gleich '1', wenn an beiden Biteingängen eine '1' ansteht. Die EXOR-Verknüpfung führt nur dann zum Ergebnis '1', wenn nur der eine Eingang '1' ist, der andere dagegen '0'. Oder: Eine '1' im Operanden führt zur Umkehrung (Invertierung) des Ergebnisses im Akku.

Wie wir sehen werden, dienen die drei logischen Befehle bei der Interfaceprogrammierung vor allem folgenden Funktionen:

ODER-Befehl:	Zuschalten eines oder mehrerer Bits.
UND-Befehl:	Maskieren, Ausschalten von Bits in den Stellen, wo das Operator-Bit '0' ist.
EXOR-Befehl:	Umschalten von Bits EIN/AUS/EIN, Flip-Flop für die Bitstellen, in denen das Operator-Bit '1' ist.

Assembler-Ausdrücke

Sofern dem Leser die Bildung von Assembler-Ausdrücken noch nicht so ganz geläufig ist, zieht er zweckmäßig einmal die Dokumentation für seinen 6502-Assembler heran. Wir erinnern: Der Assembler erlaubt die symbolische Formulierung eines Programmes als Quelltext, aus dem er ein ausführbares Maschinenprogramm generiert. Voraussetzung ist allerdings, daß im Quelltext eine genaue Erklärung der Symbole enthalten ist. Symbole dürfen wahlweise mit dezimalen, hexadezimalen, binären oder auch oktalen Werten erklärt sein, auch in der Form von Mischausdrücken aus diesen Zahlensystemen.

Ein Symbol kann weiterhin durch Gleichsetzung mit einem anderen Symbol erklärt werden. Im Beispiel

```
FIVE=     5
FUENF=    FIVE
```

haben sowohl FIVE wie FUENF den Wert 5. Für diesen Wert können in der symbolischen Programmierung beide Symbole gleichberechtigt benutzt werden.

Der Wert eines Symboles kann ferner durch Addition oder Subtraktion anderer Symbole definiert werden:

```
ZEHN=     FIVE+FIVE    oder
ZEHN=     FIVE+%101.
```

Gleiches gilt für Operanden in einem Befehl, auch sie können als Ergebnis eines Ausdruckes angesprochen werden, z.B.:

```
          LDA FIVE+5
```

Der Assembler hat also eine beschränkte Rechenhaftigkeit und kann eine Bewertung arithmetisch verknüpfter Operanden (Ausdrücke) zur Zeit der Assemblierung vornehmen. Er bildet aus Ausdrücken Direktoperanden (#) oder Adressen für das Maschinenprogramm.

Gleichsetzung per Sternanweisung

Der Assembler läßt es ferner zu, den gegenwärtigen Stand des Zuordnungszählers, der Sternadresse, einem oder mehreren Symbolen als Wert zuzuordnen. In diesem Falle spricht man von einem Label (Marke), Z.B.:

```
*=          5
FIVE
FUENF
```

Sofern nicht noch ein Befehl oder eine Anweisung wie .BYT folgt, wird vom Assembler hier nichts in den Speicher abgelegt, es handelt sich also um eine reine Erklärung mit gleicher Wirkung wie die Anweisung '='.

Die Gleichsetzung per Sternanweisung kann benutzt werden, um besonders übersichtliche Tabellen für Operatoren zu erhalten, z.B. für die Registeradressen einer VIA:

```
VIA         =$A000
            *=VIA
PORTB       =*
DDRB        =*+2
PCR         =*+$C
IFR         =*+$D
```

Symboldefinitionen für Steuerungen

Ein Steuerungssystem mag eine komplexe Menge an Ein- und Ausgabepins enthalten. Zur Übersicht für den Programmierer sollte man sich bemühen, den Ports und Pins merkfähige und ggfs. systematisierte symbolische Namen zu geben, z.B.:

Name des Ports	Namen seiner Pins
EIN1	PEIN10 bis PEIN17
AUS2	PAUS20 bis PAUS27
...	...

Je Port sollen 8 Pins individuell beobachtet oder mit definierten Ausgangssignalen versehen werden können. Die dafür benötigten 8 Binärmuster werden in gleicher Weise aber mit verschiedenen symbolischen Namen für alle Ports gebraucht. Per Assembleranweisung kann man also erklären:

```
0000                *=%1
0001 PEIN10
0001 PAUS20
0001                 *=%10
0002 PEIN11
0002 PAUS21
0002          ; ...
0002                *=%10000000
0080 PEIN17
0080 PAUS27
```

Bei dieser Art der Definition enthält das Binärmuster jedes Symboles/Operators jeweils nur eine '1', was bei der späteren Verwendung von Assembler-Ausdrücken recht zweckmäßig ist.

Wir erklären dem Assembler ferner die Adressen der benutzten Ports, z.B:

```
EIN1            =$A000
```

Systeminitialisierung

Die Ein- oder Ausgabefunktion eines Ports und seiner einzelnen Pins wird durch initialisierendes Beschreiben des Datenrichtungsregisters hergestellt. Beim VIA hat dieses Register eine um 2 höhere Adresse als der zugehörige Port. Wir initialisieren alle reinen Eingabeports z.B. wie folgt:

```
A900   LDA #0
8D02A0 STA EIN1+2      ;DATENRICHTUNG: EINGANG
8D12A0 STA EIN2+2      ;DITO
```

und entsprechend alle reinen Ausgangsports mit

```
A9FF   LDA #$FF
8D03A0 STA AUS1+2      ;DATENRICHTUNG: AUSGANG
8D13A0 STA AUS2+2
```

Nach dieser Funktionsherstellung wird es die nächste Aufgabe sein, die vom Prozeß benötigten anfänglichen Einschaltbedingungen auf die Ausgabeports zu schreiben. Zuvor aber noch ein wichtiger Hinweis: Nach einem RESET werden die internen Register der Interfacebausteine auf 00 zurückgesetzt (außer Timer und SR), alle Ports sind damit zunächst Eingänge und es können an Ausgangsleitungen anfangs unerlaubte oder auch gefährliche Einschaltbedingungen eintreten. Man wird das durch externe Beschaltung verhindern müssen.

Die Herstellung definierter Ausgabesignale per Programm ist nach der Vorarbeit in den obenstehenden Abschnitten nun übersichtlich und einfach. Gesetzt den Fall, es sollten die Pins PAUS20 und PAUS27 ein '1.-Signal erhalten, dann programmieren wir:

```
A981   LDA #PAUS20+PAUS27
8D11A0 STA AUS2
```

Der Assembler addiert die Operatoren PAUS20 und PAUS27 zu 81, und wir sehen den Vorteil der Verfahrensweise: Jeder Operator bildet nur einen Pin mit einer binären '1' ab. Aus der Addition der Symbole tritt damit kein Übertrag in eine benachbarte Bitstelle auf. Bei entsprechender Namensvergabe bleibt die Programmierung sinnfällig, und man muß nicht erst in Schaltplänen nachschauen, um die binäre Beschickung eines Ports zu programmieren.

Zuschalten von Ausgangssignalen

Für das Hinzuschalten von Ausgangssignalen mit '1' erkannten wir das logische ODER für geeignet. Also:

```
AD01A0 LDA AUS1        ;LADE ALTE AUSGABE
0942   ORA #PAUS11+PAUS16   ;ZUSCHALTEN
8D01A0 STA AUS1        ;NEUE AUSGABE AUF DEN PORT
```

Der Assembler berechnet PAUS11+PAUS16 zum Direktoperanden %01000010= $42 im Befehl. Per Programm senden die Pins 1 und 6 am Port zusätzlich auf HIGH.

Abschalten von Ausgangssignalen

Ein Abschalten erfolgt per Maskieren mit logisch UND. Alle nicht abzuschaltenden Pins sollen unverändert bleiben. Wir führen daher das Symbol ein

```
ALL=        %11111111
```

Abschalten z.B. von PAUS16 erfolgt dann mit

```
AD01A0 LDA AUS1        ;ALTES AUSGABESIGNAL
29BF   AND #ALL-PAUS16  ;MASKIEREN
8D01A0 STA AUS1        ;NEUES AUSGABESIGNAL
```

Der Direktoperand des AND-Befehles wird vom Assembler zu %10111111 berechnet, entsprechend $BF, der das Bit 6 maskiert.

Umschalten von Ausgangssignalen

Der Pin PAUS16 soll in der Art eines Flip-Flops umgeschaltet werden:

```
AD01A0 LDA AUS1        ;ALTES AUSGABESIGNAL
4940   EOR #PAUS16     ;UMSCHALTEN
8D01A0 STA AUS1        ;NEUES SIGNAL IN DEN PORT
```

Ein weiteres Beispiel: Alle übrigen Ausgangssignale eines PORT1 sollen umgeschaltet werden, nur die von PAUS11 und von PAUS16 nicht:

```
AD01A0 LDA AUS1
49BD   EOR #ALL-PAUS16-PAUS11   ;UMSCHALTEN
8D01A0 STA AUS1
```

In vielen Fällen wird man statt eines mehrteiligen Ausdruckes sicher ein zusammenfassendes Symbol erklären, um ein 'sowohl als auch' abzudecken. Bei allen Berechnungen auf Assemblerebene beachte man jedoch, daß keine Überträge in benachbarte Bitstellen eintreten dürfen.

Die Erzeugung von Ausgabesignalen in symbolischer Programmierung dürfte damit ausreichend erklärt sein. Jetzt sind Vorschläge zu machen, wie man Eingabesignale analysiert. Für Prüfungen per Software bieten sich vor allem die Befehle BIT und CMP an, denn sie zerstören keine Registerinhalte und liefern nur einen Status ab. Der BIT-Befehl transportiert die beiden vordersten Bits des Operanden in das N- und V-Flag des Status.Zugleich führt er aber auch ohne Zerstörung des Akkus ein logisches AND Akku zu Operand durch und setzt das Z-Flag in Abhängigkeit vom gedachten AND–Ergebnis.

Prüfung eines Eingabesignales

Die folgenden Beispiele benutzen die AND-Seite des BIT-Befehles, nicht jedoch die besonderen Fähigkeiten, das N- und V-Flag zu beeinflussen. Der Operator ist vor dem BIT-Vergleich in den Akku zu laden, z.B. warten, bis Bit PEIN12 'HIGH' ist.

```
        A904   LDA #PEIN12     ;DIREKTOPERATOR
LOOP    2C00A0 BIT EIN1        ;PRUEFUNG DES PORTS
        F0FB   BEQ LOOP
```

Prüfung mehrerer Eingabesignale an einem Port

Das vorliegende Beispiel läßt sich einfach auf die Prüfung mehrerer Bit erweitern, ob alle z.B. LOW sind:

```
        A91C    LDA #PEIN12+PEIN13+PEIN14
        2C00A0  BIT EIN1
        F005    BEQ *+7         ;VERZWEIGEN, WENN ALLE 3 EINGAENGE LOW
        ;.. WEITER, SOBALD  AUCH    ;NUR EINER DER EINGAENGE HIGH
```

Die kombinierte Abfrage mehrerer Pins wird nicht immer darauf erfolgen, daß alle betrachteten Eingangssignale gleich sind, entweder alle'0' oder alle'1'. In solchen Fällen müssen wir aus der Gesamtinformation eines Eingangsports eine Teilmenge isolieren, die den interessierenden Pins entspricht. Das geschieht mit dem AND-Befehl. Anschließend wird mit dem CMP-Befehl das Bitmuster der Bedingung abgefragt. Ein Beispiel: Die Warteschleife darf nur verlassen werden, wenn PEIN12 und PEI13 HIGH sind und wenn zugleich PEIN14 LOW ist:

```
LOOP1   AD00A0  LDA EIN1         ;LADE EINGANGSPORT
        291C    AND #PEIN12+PEIN13+PEIN14   ;ISOLIERE INFORMATION AN DEN 3 PINS
        C90C    CMP #PEIN12+PEIN13
        D0F7    BNE LOOP1
```

Prüfung der Eingabesignale mehrerer Ports

In größeren Systemen mögen die eine Bedingung bildenden Eingangssignale an verschiedenen Ports anliegen. In diesem Falle durchläuft man eine Abfragekette. Im Beispiel soll die Schleife nur verlassen werden, wenn PEIN13=HIGH und zugleich PEIN22=LOW:

```
LOOP2
        A908    LDA #PEIN13     ;DIREKTOPERAND
        2C00A0  BIT EIN1        ;PORTPRUEFUNG
        F0F9    BEQ LOOP2       ;PEIN13 NOCH LOW
        A904    LDA #PEIN22     ;DIREKTOPERAND
        2C10A0  BIT EIN2        ;PORTPRUEFUNG
        D0F2    BNE LOOP2       ;PEIN22 NOCH HIGH
```

Bei dieser Art der Abfrage verläßt man die Schleife nicht immer, wenn nämlich die erwartete Signalkombination nur für kurze Mikrosekunden stabil ist. In diesen Fällen ist zusätzliche Außenbeschaltung zu empfehlen.

Prüfung alternativer Eingangsbedingungen

Gelegentlich ist eine Aktion einzuleiten, wenn die eine oder die andere Signalkombination an Eingangspins vorgefunden wird. In diesem Falle arbeitet man wie oben mit dem CMP-Befehl, schaltet jedoch mehrere Abfragen hintereinander und verläßt die Abfrageschleife jeweils mit BEQ.

Prüfung mehrheitlicher Bedingungen

In verschiedenen Systemen soll eine Aktion erst dann erfolgen, wenn eine Mindestzahl von Eingabebedingungen wahr ist. So sei z.B. gefordert, daß von den vier SignalenPEIN14 bis PEIN17 mindestens 3 auf '1' sind:

```
PRUEF
        A204    LDX #4          ;SCHLEIFENZAEHLER
        A000    LDY #0          ;ZAEHLT ERFUELLTE BEDINGUNGEN
        AD00A0  LDA EIN1        ;PORTINHALT
```

```
        29F0    AND #PEIN14+PEIN15+PEIN16+PEIN17   ;MASKIERUNG FUER ERWEITERTE
PR1     0A      ASL A              ;BIT IN'S CARRY ROLLEN                FAELLE
        9001    BCC PR2            ;BIT WAR 0
        C8      INY                ;BEDINGUNGSZAEHLER +1
PR2     CA      DEX                ;SCHLEIFENZAEHLER
        D0F9    BNE PR1            ;MEHR LOOPINGS
        C003    CPY #3             ; 3 X BEDINGUNG ZUTREFFEND?
        90EC    BCC PRUEF          ;NEIN, NOCH NICHT
```

Zusammenfassung

Mit dem Assembler programmiert man so nahe an den Fähigkeiten der Maschine wie möglich und mit den zumeist gewünschten kürzesten Ausführungszeiten.

Eine höhere Sprache wie BASIC fördert nicht die für die Interfaceprogrammierung notwendige gedankliche Klarheit. Kann man sich unter POKE49961,23 vorstellen, daß z.B. am Port A des AIM die Bits 3 und 4 auf '1' gesetzt werden?

Die in diesem Artikel enthaltenen Vorschläge zur Definition von Pins und die Verknüpfung ihrer Symbole in Assembler-Ausdrücken mögen dem Programmierer bei der Problemlösung für komplexe Schaltungen helfen. Im Laufe der Zeit werden Verbesserungsvorschläge aus der Leserschaft interessieren und auch Vergleiche hinsichtlich des Einsatzes sog. Steuersprachen.

R. L.

¤

AIM 65 FORTH

Die bereits angekündigte Sprache FORTH wurde von Rockwell auf der electronica 80 vorgestellt, allerdings noch in EPROM-Version, weil die maskenprogrammierten ROMs noch nicht fertig geworden waren. Lieferfähigkeit für die 2 ROMs mit zus. 8 KB und das dazugehörige Benutzerhandbuch soll aber noch in diesem Jahre bestehen.

FORTH wurde im August-Heft (8/80) der Zeitschrift BYTE ausführlich vorgestellt. Diese Sprache ist wie folgt zu charakterisieren: Die ROMs enthalten in ihrem dictionary über 200 Kernfunktionen. Der Benutzer kann neue Funktionen mit Namen kompilieren, indem er neue Konstrukte auf die älteren aufbaut. Diese werden dann ebenso Teil des dictionary. Die Sprache ist stack-orientiert, d.h. Parameter werden in umgekehrter polnischer Notation (UPN) über den Stack übergeben. Es handelt sich um eine 'threaded language' (aufgefädelt) mit kompakter Speicherausnutzung in Form von Tabellen aufzurufender Unterprogramme. Obwohl im Kern interpretativ, führt FORTH zu einer wesentlich schnelleren Ausführung (ca. 10x) als andere Interpreter und kann in manchen Fällen an Maschinenprogrammierung heranreichen. Das wird erreicht durch die problemlose Einbindung von Assembler-Programmierung. Variable können in beliebigen Zahlensystemen zwischen 2 und 40 als Zahlenbasis definiert sein.

FORTH ist wohl weniger eine Sprache für vorwiegend zahlrenrechnende Anwendungen aber sehr geeignet für das Steuern und Messen, Testautomaten und Datenerfassung. Sie wird damit für Applikationen benutzt werden, die bisher der Assembler-Programmierung vorbehalten waren. - Diese Zeitschrift wird frühestmöglich auf die FORTH-Programmierung eingehen und auch entsprechende Artikel aus der Leserschaft veröffentlichen.

Zur ersten Orientierung ein Abdruck des Befehlssatzes aus dem Datenblatt von Rockwell:

AIM 65 FORTH Words

ni = 16-bit signed number on the stack (n1 is the deepest value).
di = 32-bit signed number on the stack (d1 is the deepest value).
ui = 16-bit unsigned number on the stack (u1 is the deepest value).

STACK MANIPULATION

DUP	Duplicate top of stack.
2DUP	Duplicate top two stack items.
DROP	Delete top of stack.
2DROP	Delete top two stack items.
SWAP	Exchange top two stack items.
OVER	Copy second item to top.
ROT	Rotate third item to top.
-DUP	Duplicate only if non-zero.
>R	Move top item to return stack.
R>	Retrieve item from return stack.
R	Copy top of return stack onto stack.
PICK	Copy the nth item to top.
SP@	Return address of stack position.
RP@	Return address of return stack pointer.
SO	Return address of pointer to bottom of stack.
BOUNDS	Convert "address count" to "end-address start-address"
.S	Print contents of stack.

NUMERIC REPRESENTATION

DECIMAL	Set decimal base.
HEX	Set hexadecimal base.
BASE	Set number base.
DIGIT	Convert ASCII to binary.
0	The number zero.
1	The number one.
2	The number two.
3	The number three.

COMPARISON OPERATORS

<	True if n1 less than n2.
>	True if n1 greater than n2.
=	True if top two numbers are equal.
0<	True if top number negative.
0=	True if top number zero.
U<	True if u1 less than u2.
NOT	Same as 0=.

ARITHMETIC AND LOGICAL

+	Add.
D+	Add double-precision numbers.
-	Subtract (n1 - n2).
*	Multiply.
/	Divide (n1/n2).
MOD	Modulo (i.e. remainder from division).
/MOD	Divide, giving remainder and quotient.
*/MOD	Multiply, then divide (n1*n2/n3), with double intermediate.
*/	Like */MOD, but give quotient only.
U*	Unsigned multiply leaving double product.
U/	Unsigned divide.
M	Signed multiplication leaving double product.
M/	Signed remainder and quotient from double dividend.
M/MOD	Unsigned divide leaving double quotient and remainder from double dividend and single divisor.
MAX	Maximum.
MIN	Minimum.
+-	Set sign.
D+-	Set sign of double-precision number.
ABS	Absolute value.
DABS	Absolute value of double-precision number.
NEGATE	Change sign.
DNEGATE	Change sign of double-precision number.
S - > D	Sign extend to double-precision number.
1+	Increment value on top of stack by 1.
2+	Increment value on top of stack by 2.
1-	Decrement value on top of stack by 1.
2-	Decrement value on top of stack by 2.
AND	Logical AND (bitwise).
OR	Logical OR (bitwise).
XOR	Logical exclusive OR (bitwise).

CONTROL STRUCTURES

DO . . . LOOP	Set up loop, given index range.
DO . . . +LOOP	Like DO . . . LOOP, but adds stack value to index.
I	Place current index value on stack.
LEAVE	Terminate loop at next LOOP or +LOOP.
BEGIN . . . UNTIL	Loop back to BEGIN until true at UNTIL.
BEGIN . . . WHILE . . . REPEAT	Loop while true at WHILE; REPEAT loops unconditionally to BEGIN.
BEGIN . . . AGAIN	Unconditional loop.
IF . . . THEN	If top of stack true, execute following clause THEN continue; otherwise continue at THEN.
IF . . . ELSE . . . THEN	If top of stack true, execute ELSE clause THEN continue; otherwise execute following clause, THEN continue.

OUTPUT FORMATTING

NUMBER	Convert string at address to double-precision number.
<#	Start output string.
#	Convert next digit of double-precision numbe and add character to output string.
#S	Convert all significant digits of double-precision number to output string.
SIGN	Insert sign of n into output string.
#>	Terminate output string (ready for TYPE).
HOLD	Insert ASCII character into output string.
HLD	Hold pointer, user variable.
-TRAILING	Suppress trailing blanks.
.LINE	Display line of text from mass storage.
COUNT	Change length of byte string to type form.
.	Print number on top of stack.
.R	Print number n1 right justified n2 places.
D.	Print double-precision number n2 n1.
D.R.	Print double-precision number n2 n1 right justified n3 places.
DPL	Number of digits to the right of decimal point

INPUT-OUTPUT

CR	Carriage return.
SPACE	Type one space.
SPACES	Type n spaces.
." ....."	Print text string (terminated by ").
DUMP	Dump n2 words starting at address.
TYPE	Type string of n1 characters starting at address n2.
?TERMINAL	True if terminal break request present.
KEY	Read key, put ASCII value on stack.
EMIT	Output ASCII value from stack.
EXPECT	Read n1 characters from input to address n2.
WORD	Read one word from input stream, until delimiter.
IN	User variable contained within TIB.
HOLD	Waits for KEY.
BAUD	Set BAUD rate.
BL	Output a SPACE character.
C/L	Number of characters/line.
TIB	Pointer to terminal input buffer start address.
B/SCR	Number of blocks/editing screen.
QUERY	Input text from terminal.
ID.	Print <name> from name # field address (nfa).

MEMORY

@	Fetch value addressed by top of stack.
!	Store n1 at address n2.
C@	Fetch one byte only.
C!	Store one byte only.
?	Print contents of address.
+!	Add second number on stack to contents of address on top.
CMOVE	Move n3 bytes starting at address n1 to area starting at address n2.
FILL	Put byte n3 into n2 bytes starting at address n1.
ERASE	Fill n2 bytes in memory with zeroes, beginning at address n1.
BLANKS	Fill n2 bytes in memory with blanks, beginning at address n1.
TOGGLE	Mask memory with bit pattern.

MISCELLANEOUS AND SYSTEM

(<comment>)	Begin comment (terminate by right parentheses on same line).
IN	System variable containing offset into input buffer.
LIMIT	Top of memory.
QUIT	Clear return stack and return to terminal.

COMPILER-TEXT INTERPRETER

—>	Interpret next screen.
;S	Stop interpretation.
COMPILE	Compile following <name> into dictionary.
LITERAL	Compile a number into a literal.
DLITERAL	Compile a double-precision number into a literal.
EXECUTE	Execute the definition on top of stack.

DICTIONARY CONTROL

FORGET	FORGET all definitions from <name> on.
HERE	Returns address of next unused byte in the dictionary.
ALLOT	Leave a gap of n bytes in the dictionary.
TASK	A dictionary marker.
'	Find the address of <name> in the dictionary.
-FIND	Search dictionary for <name>.
DP	User variable containing the dictionary pointer.
C,	Store byte into dictionary.
,	Compile a number into the dictionary.
PAD	Pointer to temporary buffer.

MONITOR & CASSETTE I/O

COLD	AIM 65 FORTH cold start.
MON	Exit to AIM 65 Monitor.
?TTY	Switch: true = TTY; false = KB.
CHAIN	Chain tape file.
CLOSE	Close tape file.
?IN	Set to active input device (AID).
?OUT	Set to active output device (AOD).
GET	Input a character from the AID.
PUT	Output a character to the AOD.
READ	Input n2 characters from AID to address n'
WRITE	Output n2 characters to AOD at address n
SOURCE	Compile from the AID.
FINIS	Terminate compile from SOURCE.
-CR	Output CR to printer only.

VIRTUAL STORAGE

LOAD	Load mass storage screen (compile or execute).
BLOCK	Read mass storage block to memory addr
B/BUF	System constant giving mass storage bloc size in bytes.
BLK	System variable containing current block number.
SCR	System variable containing current screen number.
UPDATE	Mark last buffer accessed as updated.
FLUSH	Write all updated buffers to mass storage.
EMPTY-BUFFERS	Erase all buffers.
+BUF	Increment buffer address.
BUFFER	Fetch next memory buffer.
R/W	User read/write linkage.
USE	Variable containing address of next buffer.
FIRST	Leaves address of first block buffer.
OFFSET	User variable block offset to mass storage
PREV	Variable containing address of latest buffe

VOCABULARIES

CONTEXT	Returns address of pointer to CONTEXT vocabulary.
CURRENT	Returns address of pointer to CURRENT vocabulary.
FORTH	Main FORTH vocabulary.
ASSEMBLER	Assembler vocabulary.
VOCABULARY <name>	Create new vocabulary.
VLIST	Print names of all words in CONTEXT vocabulary.
VOC-LINK	Most recently defined vocabulary.

SECURITY

ABORT	Error; operation terminates.
ERROR	Execute error notification and restart syste
MESSAGE	Displays message.
WARNING	Pointer to message routine.
FENCE	Prevents forgetting below this point.
WIDTH	Controls significant characters of <name>

PRIMITIVES

OBRANCH	Run-time conditional branch.
BRANCH	Run-time unconditional branch.
PUT	Stores registers and jumps to next.
ENCLOSE	Text scanning primitive used by WORD
RO	Location of return Stack.
RP!	Initializes return Stack.
SO	Initial value of stack pointer.
SP!	Initialize stack pointer.
NEXT	The FORTH virtual machine.

Fortsetzung auf Seite 32

Der AIM 65/40

Auf der electronica 80 in München präsentierte Rockwell den 'professional microcomputer' AIM 65/40. Es handelt sich um ein insgesamt neu entworfenes System, das sowohl von der Hardware als auch vom Betriebsprogramm her in neue Leistungsbereiche vordringt, allerdings auch in neue Preisbereiche (ca. DM 4.000,-). Die Serienfertigung ist für Mai/Juni 1981 angekündigt.

Eine erste Charakterisierung: Computer mit 12K Monitor/Editor, Steckplätzen für weitere 20K Festwertspeicher sowie Bestückungsmöglichkeit bis 48K RAM auf der Platine (grundausstattung offensichtlich 16K), Schreibschutz ist in Blöcken von 8K möglich. Per Programm kann auf eine zweite Speicherbank umgeschaltet werden, so daß insgesamt 131 K Bytes adressierbar sind.

Die Systemeinheiten Fluoreszenz-Display und Thermodrucker sind für eine Zeilenbreite von 40 Zeichen eingerichtet und sind jeweils mit einer eigenen CPU versehen. Als 'intelligente' Ausgabegeräte entlasten sie die CPU der Hauptplatine, und sie können auch bis zu 1,80 m von ihr entfernt betrieben werden.

Der Thermodrucker kennt zwei Betriebsarten, nämlich Zeichen/Grafik. In der ersteren erzeugt er in 7x9 Punktmatrix 256 Zeichen, nämlich Ziffern, Buchstaben (groß und klein), griechische Buchstaben, deutsche Umlaute, hoch- und tiefgestellte Ziffern (Hochzahlen, Indices), mathematische und Sonderzeichen sowie semigrafischeZeichen, die für die Erzeugung von Diagrammen etc. geeignet sind. Im Grafik-Modus wird je Punktzeile eine Auflösung von 280 Punkten erzielt. Im Zeichenmodus beträgt die Druckleistung 240 Zeilen/Min..

Das helle grüne Fluoreszenz-Display formt die Zeichen aus 16 Segmenten. Es werden der Zeichensatz des ASCII uppercase (Großbuchstaben, Ziffern) sowie spezielle semi-grafische Zeichen abgebildet. - Die Tastatur ist auf 57 Tasten erweitert (gegenüber AIM 65), mit Umschaltung aller Tasten (shift lock), ATTN, RESET und 8 anwenderdefinierten Funktionstasten.

Die Hauptplatine hat 4 Kontaktleisten mit anderen Zungenabständen und Belegungen gegenüber AIM 65. Eine 40-polige Leiste führt eine 6522 Anwender-VIA nach außen, ein 26-Pin serieller Anschluß dient einem RS-232C-Interface (mit 6551 ACIA), über einen Anschluß mit 20 Pin sind Interfaces für 2 Cassettenrecorder und TTY (20 mA) herausgeführt. Die Systembusse sind auf eine 72-polige Leiste gelegt, wobei Kompatibilität mit den Microflex 65-Anschlüssen besteht. Eine Beschaltung auf der Hauptplatine erlaubt die Zuordnung von Prioritäten zu Interruptereignissen.

Bei dem auf der Messe vorgeführten Prototyp waren die Flächenmaße der AIM-Hauptplatine noch nicht eingehalten, sie werden aber angestrebt. - Stromversorgung und Gehäuse sind auch weiterhin vom Benutzer vorzuhalten.

Das Betriebssystem, bisher in EPROMs vorgeführt, soll 12 KB umfassen. Die Leistungen des AIM 65-Monitors waren in 8 KB enthalten. Davon sind rein rechnerisch erhebliche Abzüge zu machen, weil Drucker und Display verselbständigt werden. Netto dürften die Leistungen des Betriebssystems um etwa 5 KB zunehmen. Die nachfolgende Liste der erweiterten Monitor- und Editor-Kommandos läßt nichts erkennen, was soviel Maschinenprogramm erfordern würde. Insofern darf man nach der Freigabe noch einige Überraschungen erwarten. Verbesserungen liegen ganz offensichtlich im Editor, der auch mit Zeilennummern arbeiten kann.

Den Gesprächen mit den Herren von Rockwell/Kalifornien war zu entnehmen, daß die Zusatzprogramme Assembler und BASIC für den AIM 65/40 nicht neu bearbeitet werden sollen. Andererseits soll für den neuen Computer die Ein- und Ausgabe RAM-vektoriert werden, was neueren Software-Philosophien entspräche. Diese Vektorierung würde aber für die vorhandenen ROMs mit ihren Zugriffen auf die AIM 65-E/A Schwierigkeiten bedeuten. Bei beabsichtigter Aufwärts-Kompatibilität sollte dieser unaufgeklärte Punkt beachtet werden.

Der bisherige AIM 65 ist, auch in der Form des Siemens PC 100, in vielen Betrieben bisher schon als Entwicklungssystem eingesetzt worden, und zwar unter Verzicht auf das größere System 65 von Rockwell. Der neue AIM 65/40 stärkt die Mitte dazwischen. Der Anwender erhält leistungsfähige Betriebs- und Sprachprogramme und die Möglichkeit, ohne Lötarbeiten einen umfangreichen Ausbau seines Systems vorzunehmen, wobei die speziellen Dienstleistungen des Microflex-Programmes stecker-kompatibel eingesetzt werden können.

Einer Rockwell-Aktennotiz ist zu entnehmen, daß das Design nicht nur für Entwicklungssysteme erfolgte, sondern auch für Tischrechner und OEM-Controller in Schaltschränken. Die Verkaufsphilosophie hinsichtlich der Grundausrüstung und der einen Mehrpreis bedingenden Optionen ist noch nicht ganz klar geworden. Das Betriebssystem unterstützt offensichtlich CRT-Bildschirmmonitore. Das nicht eben billige Fluoreszenz-Display mag angesichts dessen zu einer Option werden, ebenso der Thermaldrucker, denn häufig wird auf andere Geräte gedruckt.

Nachfolgend eine Liste der für Monitor und Editor vorgesehenen Kommandos.

INTERACTIVE MONITOR COMMANDS

Monitor Control Commands

CTRL RESET	Enter & Initialize Monitor (Cold Reset)
RESET	Re-Enter Monitor (Warm Reset)
ATTN	Re-Enter Monitor
ESC	Return to Monitor
E	Enter & Initialize Text Editor
T	Re-Enter Text Editor
N	Enter Optional Assembler
5	Enter Optional Compiler/Interpreter
6	Re-Enter Option Compiler/Interpreter
+	Repeat Last Command

Display/Alter Registers

R	Display Registers
A	Alter Accumulator
P	Alter Processor Status
S	Alter Stack Pointer
X	Alter X Register
Y	Alter Y Register
'	Alter Program Counter
J	Show Register Header

Display/Alter Memory

M	Display Memory at address in Hex & ASCII
SPACE	Display Next 8 Memory Locations
–	Display Last 8 Memory Locations
/	Alter Current Memory Contents

Instruction Entry/Disassembly

I	Instruction Mnemonic Entry
K	Disassemble Memory into Instructions

Execution/Trace

G	Execute Program
V	Toggle Register Trace
Z	Toggle Instruction Trace
H	Display Program Counter History

Breakpoints

?	Display Breakpoints
#	Clear Breakpoints
4	Toggle Breakpoints Enable/Disable
B	Set Breakpoint Address

Load/Dump Memory

L	Load Object Code
D	Dump Object Code
F	Verify Object Code

Peripheral Control

1	Toggle Audio Tape 1 Control On/Off
2	Toggle Audio Tape 2 Control On/Off
3	Verify Tape Checksum

Miscellaneous Commands

O	Toggle Memory Bank Select
Q	Enter Command String Interpreter
CTRL P	Toggle Printer Audit Trail On/Off
	Comment Line
:	Enter AIM 65/40 Self Test
F1–F8	Enter Function 1–Function 8

Fortsetzung auf Seite 32

Ein- und Ausgabe am AIM 65 (3)

2.3 Die Ausgabe auf den Thermodrucker

Der Betrieb des Druckers wird beim AIM 65 voll vom Monitorprogramm kontrolliert, und zwar nicht nur hinsichtlich der Beschickung und Verwaltung des Druckpuffers IBUFM ab Adresse $A460, sondern auch bezüglich der Zeichenerzeugung aus einer im ROM enthaltenen Tabelle (character generator) und hinsichtlich der koordinierten Bewegung von Druckkopf und Schreibwalze sowie des Aufheizens der Thermoköpfe zum Druckzeitpunkt.

Bei der Vielzahl der Tätigkeiten ist es nicht verwunderlich, wenn die rein physischen Ausgaberoutinen und die Zeichentabellen einen erheblichen Umfang haben. - Der Programmierer braucht im Normalfall jedoch nur auf die Druckroutine OUTPRI in $F000 zurückzugreifen. Dem Drucker ist weiterhin ein Ausgabepuffer IBUFM zugeordnet (20 Stellen ab $A460), der mit der 41. bis 60. Stelle des Display-Buffers identisch ist. Zur Verwaltung des Druckpuffers sind zugeordnet die CURPOS in $A416 (Cursor-Position) zur Bezeichnung der nächstfreien Ablagestelle sowie das Printerflag PRIFLG in $A411.

Mit dem PRIFLG hat es folgende Bewandnis: Sein Vorzeichen (vorderstes Bit) wird immer dann geprüft, wenn eine Druckzeile abzuschließen ist. Ist das Vorzeichen negativ, so erfolgt die physische Ausgabe auf das Papier, bei positiv unterbleibt sie. Die Ablage eines Zeichens im Drucker-Puffer erfolgt immer, unabhängig von diesem Vorzeichen.

OUTPRI ist im normalen Rechnerbetrieb Sprungziel der Ausgabeverteiler WHEREO und OUTALL, sofern OUT=P. Für diesen Fall wird der Drucker zwangsweise eingeschaltet und anschließend im zuvor übernommenen Einschaltzustand zurückgegeben. Wie dem Blockbild bei Abschnitt 2.2 zu entnehmen ist, wird OUTPRI aber auch angesprungen, wenn OUT=CR, d.h. wenn die Ausgabe auf die aktiven Systemeinheiten erfolgen soll. In diesem Falle erfolgt der Druck auf das Papier nur, wenn der Drucker eingeschaltet ist.

OUTPRI ist ähnlich aufgebaut wie OUTDIS:

a) Das zu druckende Zeichen wird im Akku als Parameter übergeben
b) Die Register A, X und Y werden unversehrt zurückgegeben
c) Ein 'CR' führt zur Auffüllung des Druck-Puffers ab CURPOS mit Leerzeichen und zum Ausdruck des Pufferinhaltes (20 Zeichen) in eine Zeile, wenn PRIFLG negativ. - In jedem Falle wird anschließend der gesamte Buffer mit Leerzeichen gefüllt (Blanks) und die CURPOS auf 0 zurückgesetzt.
d) Sonstige Zeichen werden an der durch CURPOS bezeichneten Stelle im Puffer abgelegt. Anschließend:
CURPOS=CURPOS+1
e) Wenn bei der Übernahme eines neuen Zeichens CURPOS=20, dann erfolgt der Ausdruck der Zeile, weil der Buffer gefüllt ist (in Abhängigkeit von PRIFLG), Rücksetzen CURPOS=0, Füllen des gesamten Buffers mit Leerzeichen und Ablage des neu angekommenen Zeichens in vorderster Bitposition.
f) Jedes 'positive' ASCII-Zeichen (Bit7=0) löscht alle Stellen rechts von seinem Bufferplatz zu Leerzeichen. Die dabei benutzte Löschroutine ist OUTPR in $F038.

Der in f) aufgezeichnete Löschmechanismus hat ein Gegenstück in OUTDIS. Wir besprachen es beim Beispiel 2.2.5 'Beschleunigte Ausgabe'. Auch hier können wir feststellen: das Löschen aller rechts vom Zeichen stehenden Buffer-Bytes ist in allen Fällen überflüssig, wo abgelegte Zeichen nicht mehr einer Korrektur durch DELETE unterliegen.

Im nachfolgenden Beispiel wollen wir zeigen, daß man einen im Druckpuffer enthaltenen Standard-Text mehrfach ausdrucken und sogar von Zeile zu Zeile abändern kann.

Beispiel 2.3.1: Mehrfacher Ausdruck eines Textes mit Änderung vom Keyboard.

```
                PRINT INTO SAME LINE
                ;--------------------
                ;SYMBOLS
OUTPRI          =$F000
CURPOS          =$A416
PRIFLG          =$A411
IPST            =$F045
GETKEY          =$EC40
CR              =$D
                ;--------------------
                ;MAIN PROGRAM
                *=$200
0200  A90D      LDA #CR         ;CLEAR
0202  2000F0    JSR OUTPRI      ;PRINTER BUFFER
0205  A200      LDX #0          ;COUNTER & ADDRESSER
0207  BD2B02    LDA TAB,X       ;WRITE A LINE FROM TABLE
020A  2000F0    JSR OUTPRI      ;TO BUFFER
020D  E8        INX
020E  E011      CPX #TX-TAB     ;COMPARE FOR END
0210  D0F5      BNE LOOP        ;MORE CHARACTERS
0212  A20C      LDX #$C         ;ALWAYS WRITE
0214  8E16A4    STX CURPOS      ;TO SAME POSITION
0217  2040EC    JSR GETKEY      ;GET CHARACTER TO BE WRITTEN
021A  0980      ORA #$80        ;BUT DO NOT ERASE
021C  2000F0    JSR OUTPRI      ;TO END OF BUFFER
021F  38        SEC
0220  6E11A4    ROR PRIFLG      ;PRINTER 'ON'
0223  2045F0    JSR IPST        ;PHYSICAL WRITING
0226  2E11A4    ROL PRIFLG      ;PRINTER 'OFF'
0229  D0E7      BNE GET         ;LET'S DO IT AGAIN
022B
                ;--------------------
                ;STRING CONSTANT
022B  4920      .BYT 'I PRESS THE  -KEY'
023C            .END
```

```
MUSTERAUSDRUCKE:
I PRESS THE K-KEY
I PRESS THE 5-KEY
...
```

An diesem Beispiel ist bemerkenswert: Ohne den Befehl ORA #$80 findet eine Löschung rechts der Zeichenstelle im Printerbuffer statt. Der Drucker läßt sich bequem ein- und ausschalten, indem man das vorderste Bit des PRIFLG beeinflußt. - OUTPRI ist vor allem eine Transportroutine für den Printerbuffer. Der eigentliche Ausdruck wird von IPST und nachfolgenden Routinen besorgt.

Die Druckerroutinen lassen sich für spezielle Zwecke manipulieren. Weltweit bekannt wurden die Routinen des Verfassers AIM-PLOT (Plotten) und AIMGRAPH (neue Zeichensätze), erstmals abgedruckt in Heft 6 dieser Zeitschrift (nachgedruckt im Buch 1-6). - Das erste Programm ist eine Überlagerung der Schreibroutine IPO2, die nach Punktzählung bis 10 (nach einer Punktreihe) abgebrochen wird. Das zweite führt ab entsprechender Stelle wie IPS1 entspr. $F0E8 zum Laden von Punktmustern aus einem frei definierbaren character generator.

In Heft 10 wurde ferner ein Programm veröffentlicht, das den Abdruck in 60 Spalten ermöglicht. Es werden 3 Druckerstreifen erzeugt, die nebeneinander zu kleben sind.

Die in Beispiel 2.2.6 entwickelte Textausgabe aus einer Tabelle läßt sich mit geringen Änderungen auf Druckerbetrieb und natürlich auch auf andere Systemeinheiten anpassen.

Bei der vergrößerten Version des AIM 65, dem AIM 65/40, ist die gesamte Druckersteuerung auf einen spezialisierten 1-chip-Mikroprozessor übertragen, wodurch die Haupt-CPU erheblich entlastet wird. Auch das Floureszenzdisplay erhält einen eigenen Einchipper.

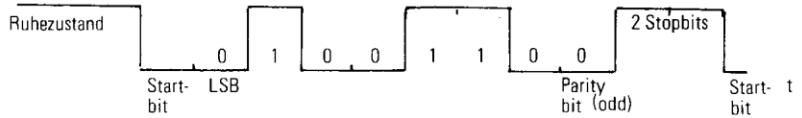
2.4 Ausgabe über TTY

Der AIM 65 hat bekanntlich ein TTY-Interface für die (nicht genormte) 20 mA-Stromschleife, und zwar ein- und ausgabeseitig. Im Monitor-Programm werden zu seiner Bedienung die Hauptroutinen GETTTY in $EBDB und OUTTTY in $EEA8, mit denen ASCII-Terminals (Fernschreiber, Video-Platinen, Drucker, andere Rechner) betrieben werden können, vorgehalten. Für Baudot-Geräte sind eigene Treiberprogramme zu schreiben.

Ein ASCII-Terminal wird zur Systemeinheit,indemman den Schiebeschalter KB/TTY auf TTY stellt. Die Eingabe wird danach von der TTY-Tastatur her erwartet, die Ausgabe erfolgt auf das Terminal. Die Tastatur des AIM und sein Printer sind bei dieser Betriebsweise inaktiviert (s.a. Blockdiagramm in 2.2). Der genannte Schiebeschalter bewirkt keinerlei hardwaremäßige Umschaltung von Signalen. Er wirkt wie ein Softwareschalter und ist es sogar auch materiell, denn er schaltet das Bit PB3 der Speicheradresse VIA Port B. - Die den Schaltzustand überprüfende Routine ist TTYTST in $E842.

Unanbhängig vom Schiebeschalter kann der Benutzer die E/A-Routinen GETTTY und OUTTTY beliebig per Betriebs- oder Anwenderprogramm verwenden.

TTY-Betrieb ist serielle Datenübertragung, d.h. ein Byte wird in einer Folge von 8 zeitlich aufeinanderfolgenden Bit gesendet. Eine solche Sendung ist für den Prozessor ein 'asynchrones' Ereignis, denn er kann ja nicht wissen, wann die Gegenseite loslegt, und die Impulswechsel sind keineswegs mit dem Maschinentakt, bes. R/W synchron. Es bedarf also weiterer Definitionen und der Möglichkeit der Synchronisierung. Für die Impulsfolge bei serieller Datenübertragung besteht daher folgendes Protokoll:



Der Ruhezustand wird also durch high level gekennzeichnet. Das Startbit (low level) enthält keine andere Information, als daß die Übertragung eines Zeichens begonnen hat. Der Empfänger erhält also sein Synchronisationszeichen. Die eigentlichen Daten folgen in den 7 folgenden Bitzeiten, gesendet wird ASCII, und zwar beginnend mit dem niederwertigsten Bit. Zur 8. Bitzeit wird entweder eine logische '0' gesandt (weil das 8. Bit bei ASCII eine '0' ist) oder aber ein Parity-Bit, das die Gesamtmenge (einschl. 8. Bit) der gesendeten 1er Bits entweder auf eine gerade Zahl (parity even) oder auf eine ungerade ergänzt (parity odd). Die Menge der auf das 8. Datenbit folgenden Bitzeiten (1-2 Stop-Bits) ist nicht genormt.

Serielle Datenübertragung erfolgt mit unterschiedlicher Geschwindigkeit. Die Baudrate gibt die Zahl möglicher Impulswechsel pro Sekunde an. Bei Verwendung nur eines Stop-Bits beansprucht die Übermittlung eines Bytes 10 Bitzeiten. Bei 300 Baud werden also 30 Zeichen

pro Sekunde übertragen. - Unabhängig von Impulsfolge und Datenübertragungsrate ist das weitere Protokoll der Datenübertragung, die Wahl des Start- und Endbytes, das Handshake während der Übertragung usw..

Die Übertragungsgeschwindigkeit wird beim AIM nicht mit einem Schalter eingestellt. Sie wird als das Ergebnis einer Zeitmessung berechnet. Entsprechend der Anweisung auf Seite 9-30 des Anwenderhandbuches sendet das Terminal dafür ein RUBOUT. AIM mißt dabei die Dauer des Startbits. Leider ist in der Teilroutine PATCH1 zur Messsung der Baudrate ein Subtraktionsfehler enthalten, der bei gewissen Geschwindigkeiten zu unverwertbaren Ergebnissen führt. Wenn die Baudrate des Terminals bekannt ist, tut man meistens gut daran, die Baudrate manuell oder per Programm in den Adressen A417/A418 zu hinterlegen, und zwar entsprechend Tabelle auf Seite 9-31 im Anwenderhandbuch.

Nach dieser Übersicht zur Routine OUTTTY in $ EEA8: Sie fügt sich nahtlos in die übrigen Ausgaberoutinen ein. Das im Akku übergebene Zeichen und die Registerinhalte von X und Y werden am Schluß wieder hergestellt. OUTTTY sendet 1 Startbit, 8 Datenbits und 2 Stopbits. Beim 8. Datenbit ist keinerlei Vorkehrung für das Senden mit parity odd oder even getroffen. Interessant ist, wie anfangs das niederwertigste Datenbit vor das Sendefenster des Ausgabepins PB2 gerollt wird (ähnliches gilt für GETTTY).

Die Festschreibung des erzeugten Ausgabesignals erfolgt für eine Bitzeit durch das Upro DELAY, in dem die Baudrate verwertet wird. Man sollte weiterhin bemerken: An OUTTTY ist zwangsweise die Routine OUTDIS (Display) angeschlossen, außer für Line Feed und das Zeichen $FF. Für viele Anwendungen ist das überflüssig, sicher auch in den Fällen, wo ein Bildschirmterminal per TTY-Schleife angeschlossen ist.

Das führt auf eine weitere Besonderheit: Die Systemeinheiten Display und Thermodrucker benötigen für den Übergang auf eine neue Zeile nur das Sonderzeichen Carriage Return. Das Betriebssystem trifft dabei Vorsorge, daß auf deren Ausgabepuffer kein LF (line Feed) trifft. Auch im Textspeicher des Editors wird am Zeilenende kein LF abgelegt. Andererseits benötigen TTY-Einheiten, wie auch andere Drucker, außer CR (Wagenrücklauf) auch ein LF zum Vorschub auf neue Zeile. Es liegt daher in der Sorgfalt des Benutzers, für solche Einheiten auf ein per Abfrage erkanntes CR ein LF per Programm nachzuschießen.

Als Beispiel für diesen Abschnitt geben wir ein Programm, das aus einem 7-Bit-ASCII-Zeichen ein 8-Bit-Zeichen mit gerader Parität erzeugt:

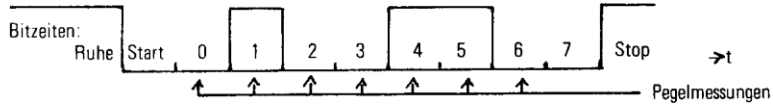
Beispiel 2.4.1: Parity-Generator

```
                    ;--------------------
                    ; PARITY FOR CHARACTER IN A
                    ;
0400                ;
0400  48      PHA   ;SAVE X & Y
0400  A207    LDX #7;SAVE 7-BIT CHARACTER
0401  A000    LDY #0;INSTALL COUNTER
0403  6A      ROR A ;START EVEN
0405  6A      ROR A ;SHIFT BIT INTO CARRY
0406  9001    BCC *+3;SKIP ON 0-BIT
0408  C8      INY   ;COUNT 1-BIT
0409  CA      DEX   ;
040A  D0F9    BNE LOOP;7 LOOPINGS
040C  98      TYA   ;GET COUNT FROM Y
040D  4A      LSR A ;GET CARRY STATUS ODD OR EVEN
040E  68      PLA   ;CHARACTER BACK
040F  9002    BCC *+4;SKIP ON EVEN COUNT
0411  0980    ORA #%10000000  ;MAKE BITS IN A EVEN   ;RESTORE X & Y
```

2.5 Dateneingabe per TTY

Abschnitt 2.4 brachte eine ausführliche Darstellung der seriellen Datenübertragung und den Hinweis, daß eine Terminal-Tastatur dann zur Systemeinheit wird, wenn der Schiebeschalter auf TTY gerichtet ist. Es ist dem Anwender unbenommen, die Empfangsroutine GETTTY in $ EBDB aufzurufen, um Daten von einem Terminal zu empfangen, auch wenn der Schalter auf KB weist.

Der Aufbau des Unterprogrammes ist einfach: Es wartet (beliebig lange) auf die abfallende Impulsflanke des Startbits. Im Anschluß daran werden 1 1/2 Bitzeiten übersprungen, und zwar durch den Aufruf von DELAY und DEHALF. Man setzt also in der Mitte der Bitzeit des ersten ankommenden Datenbits auf und bestimmt den Signalpegel High oder Low. Danach erfolgt noch sechsmal nach Ablauf jeweils einer DELAY-Zeit eine Pegelbestimmung in der Mitte der folgenden Bitzeiten. Die empfangenen Pegel werden bitweise in die Zelle CPIY eingerollt und bilden 7 Bit eines Bytes.



Während der 8. Bitzeit erfolgt keine Signalbestimmung, die Empfangsroutine macht vielmehr ab Mitte der 7. Bitzeit durch Aneinanderreihung der Aufrufe 2x DELAY und DEHALF die Augen zu, so daß erst nach der Zeit des 1. Stopbits wieder Empfangsbereitschaft besteht. Mit anderen Worten: Ein möglicherweise gesendetes parity-bit wird gar nicht erfaßt, kann also auch nicht zur Datensicherheit ausgewertet werden. Bei Datenübertragung mit erforderlicher Parity-Prüfung entwickle man daher eine eigene Routine, die 8 Datenbits erfaßt und entsprechend prüft.

2.6 Dateneingabe über Magnetbandgeräte
2.7 Datenausgabe über Magnetbandgeräte
Die Darstellung für diese Systemeinheiten erfolgt in einem späteren Heft.

3. RAM-vektorierte Ausgabe über das DILINK

Dem Blockdiagramm bei 2.2 ist zu entnehmen: Bei Stellung KB des Schiebeschalters (AIM-Tastatur und Thermaldrucker aktiv) wird eine Kette von Ausgaberoutinen durchlaufen, die in OUTDP unter $EF02 zu einem indirekten Sprung JMP (DILINK) auf das Display-Link in $ A406/07 führt. In diesen Zellen ist per normaler Systeminitialisierung der Adreßvektor von OUTDIS (EF05) enthalten.

Die Adressen A406/A407 liegen im RAM-Speicher des Bausteines 6532, wir können daher von einem RAM-vektorierten Sprung sprechen. Er wurde eingerichtet, um dem Benutzer größte Beweglichkeit bei der Gestaltung der Ausgabe zu gewährleisten. Der Benutzer kann hier per Programm (!) den Adreßvektor einer eigenen Ausgaberoutine ablegen. Eine solche Routine ist mit RTS abzuschließen. Üblich ist die Ansteuerung einer Videokarte oder eines Ausgabedruckers oder beider. Eine manuelle Änderung des Vektors führt zur Verabschiedung des Systems.

Für solche Ausgabeeinheiten wird man entsprechend den Hinweisen in 2.4 zu einem 'CR' ein 'LF' nachschießen und die Betriebsroutine anspringen. Bei TTY-verbundener Videokarte ist es OUTTTY, bei Videokarten, die über einen Port oder über die Prozessorbusse angeschlossen

sind, springt man die Durchlaufroutine ihres Betriebsprogrammes an. **Man beachte:** Man braucht eine Initialisierungsroutine zur Umsetzung des DILINK-Vektors und außerdem die erwähnte Durchlaufroutine für den Datentransport, deren Adreßvektor per Initialisierung im DILINK zu hinterlegen ist.

Es ist dem Benutzer möglich, in seiner Ausgaberoutine eine Auffächerung auf mehrere Einheiten vorzunehmen, z.B.

```
JSR VIDEO
JMP DRUCK
```

Durch die Auswertung von Kontroll-Codes (siehe Abschn. 2.1), die auch über das DILINK transportiert werden, ist es weiterhin möglich, solche Einheiten in der Ausgaberoutine ein- oder auszuschalten. Solche Vorkehrung ist insbesondere für die Assemblierung empfehlenswert. - Wenn mehrere Einheiten angeschlossen sind, dann kann man ihren jeweiligen Einschaltzustand in einem gemeinsamen Kontroll-Byte verwalten.

Für die Dateneingabe gibt es keine RAM-Vektorierung, obwohl auch sie wünschenswert wäre. Es heißt aber, daß der AIM 65/40 von solchen Möglichkeiten Gebrauch machen würde.

Wird fortgesetzt. R. L.

¤

TEXT EDITOR COMMANDS

Fortsetzung von Seite 26: Der AIM 65/40

Editor Control Commands

Q	Quit Editor and Re-enter Monitor
ESC	Return to Editor Command Level
Z	Find End of Text
+	Repeat Last Command

Line Oriented Commands

L	List Multiple Lines
R	Read Multiple Lines
I	Insert One Line
O	Overlay Current Line
K	Delete Multiple Lines
Space	Display Current Line
?	Display Current and Last Line Addresses
G	Go to Line Number
U	Go Up Multiple Lines
D	Go Down Multiple Lines
T	Go to Top Line of Text
B	Go to Bottom Line of Text
CTRL K	Go Up one Line
CTRL J	Go Down One Line

String Oriented Commands

F	Find Character String
C	Change Character String

Screen Oriented Commands

CTRL H	Go back (left) one character
CTRL L	Go forward (right) one character
CTRL X	Take Next Character
CTRL C	Clear to End of Line
CTRL D	Delete One Character
CTRL I	Toggle Insert Mode Enable/Disable
CTRL O	Clear Blink and Insert Modes
CTRL Z	Clear Display and Printer Buffers
CTRL S	Toggle Blink Mode Enable/Disable

¤

DEFINING WORDS

Fortsetzung von Seite 24: AIM 65 FORTH

: <name>	Begin colon definition of <name>.
;	End colon definition.
VARIABLE <name>	Create a variable named <name> when initial value n; returns address when executed.
CONSTANT <name>	Create a constant named <name> with value n; returns value when executed.
CODE <name>	Begin definition of assembly-language primitive operation named <name>.
.CODE	Used to create a new defining word, with execution-time "code routine" for this data type in assembly.
<BUILDS ... DOES>	Used to create a new defining word, with execution-time routine for this data type in higher-level FORTH.
CREATE	Create a dictionary header.
USER	Create a user variable.

¤

Dr. Andreas Joss, Depotstraße 16, CH-3012 Bern

Schnelles und sicheres Bandformat für AIM 65

Angeregt durch den Artikel "The Hamming-Way" von R. Löhr wurde ein schnelles Bandformat für den AIM - 65 entwickelt. Nachdem es schon beim AIM-Format immer wieder zu Leseschwierigkeiten gekommen war, wurde beschlossen, in dieser Richtung grössere Sicherheit anzustreben.

Das hier vorgestellte Bandformat wird nicht von allen Tape-Decks ohne weiteres verarbeitet. Weiter unten werden einige Hardware-Ueberlegungen gemacht und Lösungsmöglichkeiten aufgezeigt.

Das Hauptprogramm

Das Programm Fast (1067 Bytes) umfasst eine Dump- und eine Load-Routine, welche mit U-Out und U-In Vectoren aufgerufen werden. Gegenüber dem AIM-Dump hat Fast-Dump die Besonderheit, dass mit "S=" ein Sicherheitsfaktor gefragt wird. Mit S=1 wird jeder Block nur einmal übertragen (ca. 15 mal schneller als AIM-Format). Mit S=3 wird jeder Block dreimal hintereinander übertragen, immer noch ca. 5 mal schneller als AIM-Format, aber Fehlerkorrekturen sind möglich. Im Gegensatz zum Hamming-Code können hier auch grössere Fehler, z.B. durch Bandaussetzer entstanden, repariert werden.

Fast-Load braucht eine Page RAM als Input-Buffer. Eine Default-Adresse im oberen RAM Bereich wird genommen falls keine Eingabe erfolgt. Falls auf den File-Namen keine Eingabe erfolgt, wird auf dem Drucker ein Bandverzeichnis erstellt und Checksummen geprüft.

Die U-Vectoren werden nicht für jedes Zeichen einzeln durchlaufen, da dies zeitliche Probleme bieten würde. Aus demselben Grund wurden teilweise vorhandene Subroutinen nicht angesprungen um 12 μsec zu sparen.

Die Verbindung zu Editor und Basic

Zum Arbeiten mit Editor und Basic wurde das Programm "Fastlink" geschrieben (265 Bytes).

Um den Editor zu dumpen, wird nicht etwa OUT=U im Editor gegeben, sondern die Subroutine EFAST (Teil von Fastlink) aufgerufen. Der Benützer wird wie normal nach dem Sicherheitsfaktor, dem File-Namen und dem Tape gefragt. Danach wird eine File gedumpt, welche die Zero-Page Register ($DF-$E6) und den Inhalt des Editors umfasst. Nach Laden der File mit Fastload hat man den Editor auf die alten Werte initialisiert.

Basic wird ähnlich gedumpt. BFAST wird als Maschinen-Unterprogramm von Basic angesprungen. Der Benützer hat mit Beantwortung der Frage "A?" (Alles?) die Wahl mit "J" (Ja) oder "Y" (Yes) eine File zu erstellen, welche die Zero-Page Register von Basic ($00-$DE), das Programm ab $200 sowie alle Variabeln, Arrays und Strings umfasst. Falls "A?" anders beantwortet wird, werden nur Zero-Page und Programm abgespeichert.

Zum Laden wird im Monitor mit Fast-Load geladen, und anschliessend mit <6> eine Basic-Reentry gemacht. Basic steht wie vorher initialisiert zur Verfügung. Eine allfällige GWK-Expansion ist auch wieder initialisiert. Mit GOTO "Line number" kann in einem Programm weitergefahren werden, falls Variabeln etc. auch geladen wurden.

Dazu eine Bemerkung zum AIM-Basic: Nach ausgiebigen String-Manipulationen ist der String-Space, welcher zuoberst im "Used Memory" angelegt ist, teilweise beträchtlich (z.B. um 16 k Byte) angewachsen, was zu unnötig langen Files führen kann. Der Start des String-Space steht in den Adressen $7B,7C und kann durch ein Komprimieren wieder auf das Optiumum gebracht werden. Das Komprimieren geschieht durch aufrufen von FRE(0).

Erläuterungen zum Programm FAST

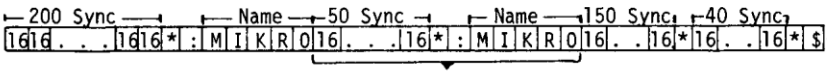
Die Zeit zwischen Uebergang von 0 auf 1 oder umgekehrt ist entscheidend: (s. auch Wechselphasenkodierung, Micromag Nr. 7, S. 12)
0 = 160 µsec, 1 = 90 µsec (Mittel 125 µsec)
d.h. die mittlere Uebertragsungsrate ist 8000 Baud, bei 4,75 cm/sec:
4300 BPI (Bit per Inch) (160 µsec = 7,6 µm Band, 90 µsec = 4,3 µm Band)

Die zu übertragenden Daten werden aufgeteilt in Blocks à 256 Byte. Falls weniger als 256 Byte übertragen werden sollen, bleibt die Blocklänge 256 Byte, es wird aber beim Laden nur der gewünschte Teil vom Input-Buffer auf Memory übertragen.

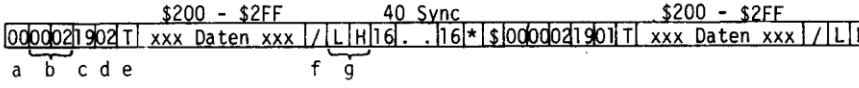
- Die Dump-Routine wird mit OUT = U aufgerufen nach <D> im Monitor und fragt zu Beginn 'S=' (Security). Jeder Block wird S mal hintereinander übertragen (auch der File-Name). Mit More wird jeweils vor dem Dump gefragt, ob noch mehr folgt (akzeptable Antworten 'Y' für Yes und 'N' für No).
- Die Load-Routine wird nach <L> im Monitor mit IN = U aufgerufen.
- Falls kein File-Name angegeben wird (CR), wird auf dem Printer ein "Directory" des Bandes gedruckt. Auf dem Display erscheint jeweils die Nummer des gelesenen Blocks und ein 'Y' oder 'N' für jeden gelesenen Block gleicher Nummer. (Y = Yes, Checksum o.k., N = No, Checksum Error). Gestopt wird mit Reset.
- Wird bei Load ein File-Name gegeben, wird mit 'B=' abgefragt, wo im RAM eine Page (256 Bytes) als Input Buffer verwendet werden kann (Startadresse). Mit 'CR' wird der Default Wert genommen und angezeigt (hier $8D00).
- Bis die richtige File erreicht wird, werden alle Files mit "SRCH F=..." angezeigt. Danach steht auf dem Display "LOAD".
- Beginnend bei Block 0 wird blockweise in den Buffer geladen. Ins Memory wird erst übertragen, wenn die Checksumme stimmte. Falls dies nicht der Fall war, wird weiter nach demselben Block gesucht. Falls beim Aufnehmen S >1 war, können so Fehler vom Band korrigiert werden. Wurde der Block erfolgreich geladen, wird nach dem Nächsthöheren gesucht. Tiefere Blocknummern werden ignoriert. Für jeden Block mit Checksum Error wird ein E auf dem Display gezeigt (nach LOAD).
- Eine Fehlermeldung erfolgt, wenn eine höhere Blocknummer als die Erwartete kam. Die Meldung kommt auf jeden Fall auf den Printer und lautet 'ERROR X'. X ist das Zeichen, das beim zweitletzten Block (welcher ja vermutlich falsch war) nach den Daten empfangen wurde, und sollte ein '/' (Slash) sein (gibt Information über die Art des Fehlers: ist es ein '/', wurde vermutlich die Decodierung verwischt (1 und 0 verwechselt), sonst wurden vermutlich Uebergänge verpasst).

Bandformat

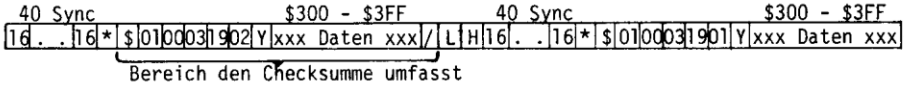
Beispiel für S = 2 und <D> von $200 - $318 sowie $10A - $10B, Name = MIKRO

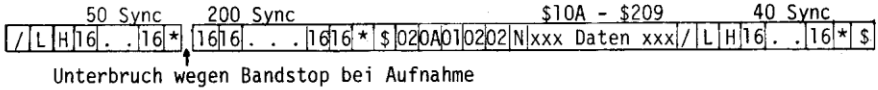


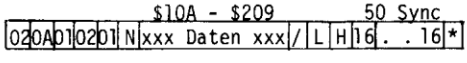
* Beendet den Sync ($16) - Strom
: Markiert einen Namenblock
$ Markiert einen Datenblock



a - Blockcount : Nummer des Blocks
b - Pointer low + high : Startadresse der Daten dieses Blocks
c - Number of Bytes : Anzahl Bytes welche aus dem Inputbuffer ins Memory verschoben werden. Falls aber 'Moreflag' = T werden alle 256 verschoben.
d - Spare Blocks : Anzahl noch folgende Wiederholungen des Blocks
e - Moreflag : Y = Yes, ja noch mehr; N = No, nein, Ende; T = temporär ja. T kann auch als total aufgefasst werden und wird verwendet, wenn alle 256 Bytes verschoben werden sollen anstatt num. of bytes
f - Slash : Schrägstrich. Verwendung bei Fehlerdiagnose
g - Checksum low + high : Prüfsumme zur Kontrolle der Datenübertragung







Das Band wird beim Laden nie angehalten. Das Umladen des Buffers nimmt etwa 5msec in Anspruch (ca Zeit von 5 Sync). 200 Sync erlauben das Starten des Tapes (ca 0.2 sec). Beim Drucken des Inhaltsverzeichnis wird das Band jeweils nach dem ersten Namenblock kurz angehalten zum Printen.

Hardware Ueberlegungen

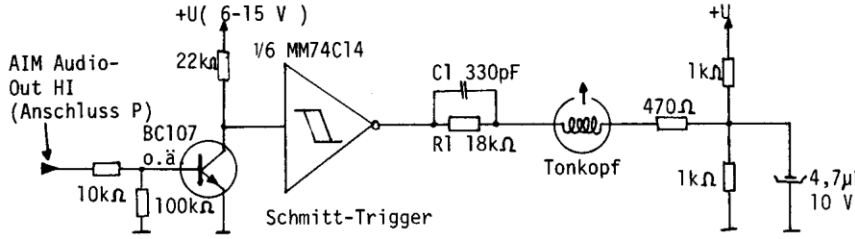
Die Elektronik herkömmlicher Kassettengeräte ist mehr zur Musikspeicherung als zur Impulsverarbeitung gedacht. Das AIM-Format trägt dem Rechnung, wie aus der Beschreibung (Users Guide, S. F-1) hervorgeht: erst die dritte der drei für den Bit-Status massgebenden Halbzyklen wird zur Decodierung verwendet, was der Elektronik genügend Zeit zum Einschwingen gibt.

Das hier vorgestellte Format, wie auch das in Micromag Nr. 6, S. 3 beschriebene Bandformat* wird von mancher Elektronik so verändert, dass anstatt kurze und lange nur noch mittellange Impulse wiedergegeben werden. Dem ist nur durch eine oder mehrere Hardwareänderungen im Bandgerät zu begegnen. Weder ein extra teures Bandgerät, dessen Elektronik nach wie vor für Musik gedacht ist, noch ein extremer Gleichlauf, der ja bei einem Zeitunterschied von fast 1:2 unkritisch ist, garantieren Abhilfe.

Die im folgenden beschriebenen Aenderungen gelten sinngemäss für alle Kassettengeräte. Die Dimensionen gelten für Philips N 2234 (ca.sFr. 110.--). Die Modifikationen sind der Wichtigkeit nach aufgeführt und brauchen nur soweit nötig durchgeführt zu werden.

1. Die Aufnahmeelektronik

Ziel ist es, das Band wechselnd in die eine oder andere Richtung magnetisch zu sättigen.
Die bei Audiogeräten übliche Vormagnetisierung kann weggelassen werden, die Hochfrequenz wird allerdings im Löschkopf weiterhin verwendet.
Die Aufnahmeelektronik wird denkbar einfach:



Die Speisespannung (6-15 Volt) wird vom Bandgerät genommen. R1 und C1 sind individuell anzupassen (Anleitung im Text).

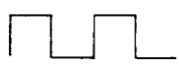
Es empfiehlt sich, auf der AIM Platine C5 zu überbrücken und C3 abzutrennen. Das Signal vom AIM wird durch den Schmitt-Trigger in ein Rechteck von 0 bis 10 V (Speisespannung des Bandgerätes, ca. 6 - 15 V möglich) umgewandelt. Der Tonkopf wird mit einem Anschluss auf halbe Speisespannung gelegt, so dass ihn entweder ein positiver oder ein negativer Strom durchfliesst. Der Widerstand R1 bestimmt im wesentlichen den fliessenden Strom. Der Kondensator C1 kompensiert die Induktivität des Tonkopfs, so dass bei richtiger Dimensionierung über dem 470Ω - Widerstand ein einwandfreies Rechteck mit einem KO (Kathodenstrahloszillograph) ge-

* Im Gegensatz zum dort beschriebenen Bandformat ist die Phasenlage bei "Fast" egal. Wenn das in Nr. 6 beschriebene Format mit der dort auf Seite 3 gezeigten Bitrepresentation laufen soll, muss entweder das Signal invertiert oder BMI und BPL in 3F5 und 3FF (Seite 9) vertauscht werden!

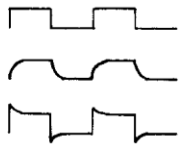
messen werden kann. Die 18 KΩ(R1) und 330 pF (C1) sind Werte, die vom Tonkopf abhängen. Bei der Wiedergabe sollte ein gut ausgesteuertes Signal (Sättigung, bestimmt durch R1) sowie kein Ueberschwingen (bestimmt durch C1) auf dem KO darzustellen sein.

Signale

am Schmitt-Trigger-Ausgang:



Spannung an 470Ω - Widerstand bei Aufnahme:



richtig

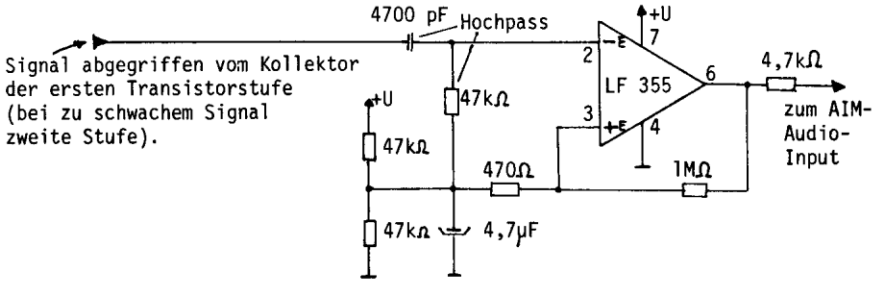
falsch, C1 zu klein

falsch, C1 zu gross

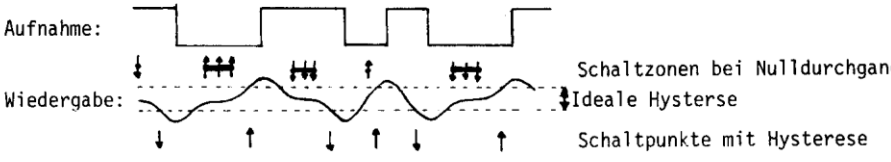
2. Die Wiedergabeelektronik

Anstatt das Signal unzählige RC-Glieder im Bandverstärker passieren zu lassen, wird das Signal schon nach der erster Vorverstärkerstufe abgegriffen und über einen Hochpass in einen Schmitt-Trigger gespiesen. Dadurch wird die Einstellung an VR1 (AIM-Platine) viel unkritischer.

Beispiel:



Dem Operationsverstärker LF355 (National) ist gegenüber dem LM307 oder LM741 der Vorteil zu geben wegen der grösseren Flankensteilheit (Slew Rate).
Mit dieser Schaltung wird immer noch der Nulldurchgang bestimmt. Nach den Erfahrungen des Autors sollte ein Betrieb spätestens jetzt möglich sein. Das Wiedergabesignal am Tonkopf sieht wie folgt aus:

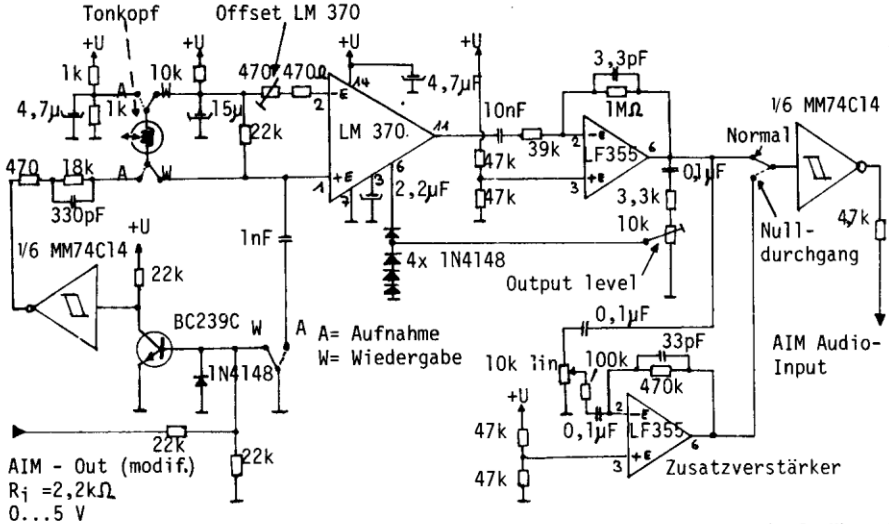


Es ist offensichtlich, dass die Nulldurchgänge nicht optimale Resultate ergeben, da sie relativ breit sind. Ideal wäre, wenn der Schmitt-Trigger eine Hysterese von ca. 1/2 des Signals hätte. Eine solche Schaltung wurde, allerdings nicht mehr mit minimalem Aufwand, realisiert. Das Signal wird vom Tonkopf fast vollständig

DC-gekoppelt (nur ein RC-Glied zur Ausschaltung von niederfrequenten Störungen) verstärkt (LM370 und LF 355). Eine automatische Verstärkungsregelung (LM 370) sorgt dafür, dass unabhängig vom Wirkungsgrad des Bandes, immer derselbe Ausgangspegel erreicht wird. Der nachgeschaltete Schmitt-Trigger (National MM 74 C14) hat eine Hysterese von 1/3 der Versorgungsspannung, somit ist die Forderung erfüllbar durch richtigen Signalpegel. Falls dennoch auf Nulldurchgang getriggert werden sollte, kann mit einem Zusatzverstärker (LF355) das Signal bis zur Uebersteuerung verstärkt werden. Diese Möglichkeit wird allerdings selten verwendet.

Der Autor betreibt 2 Kassettengeräte, von denen eines auf den beschriebenen Aufsprech- sowie den aufwendigen Wiedergabeverstärker umgerüstet wurde. Der alte Verstärker dient nur noch als Mithörkontrolle. Beim anderen Gerät wurde lediglich der beschriebene Schmitt-Trigger ergänzt. Für das AIM-Format sind beide Geräte untereinander voll kompatibel, "Fast" kann auf beiden abgespielt, aber nur auf dem einen aufgenommen werden. (Bei beiden Geräten wurde zusätzlich das Netzgerät modifiziert.)

Nachstehend die Aufnahme- und Wiedergabeelektronik des einen Bandgerätes (N.b. Die 4,7 kΩ Widerstände am Ausgang beider Wiedergabeverstärker ersparen ein Umschalten zwischen Band 1 und 2, da ja immer nur eines aktiv ist):



+U : + 10,2 V Stabilisiert (Achtung, Spannung darf während des Anfahren des Motors nicht zusammenbrechen!)

Der Nachbau obiger Schaltung setzt voraus, dass die Funktionsweise klar geworden ist, da die Dimensionierung für andere Tonköpfe teilweise anzupassen ist.

Zur Inbetriebnahme jeder dieser Schaltungen empfielt es sich, zunächst mit einem KO zu überprüfen, wie die Signale aussehen. Nachdem ein symmetrisches Rechteck einwandfrei übertragen wird, sollte man dasselbe mit einem asymmetrischen Signal, z.B. Syncstrom, versuchen. Erst dann sollte man mit dem AIM zu lesen versuchen.

```
0000
0000          ;          **********************
0000          ;          * ****************** *
0000          ;          * ****   FAST    **** *
0000          ;          * ****************** *
0000          ;          **********************
0000
0000          ; ANDREAS JOSS= DEPOTSTRASSE 16= CH-3012 BERN
0000
0000          ; THE TAPE-FORMAT "FAST" WORKS AT 8000 BAUD MEAN SPEED.
0000          ; A SPECIAL ELECTRONIC OF THE TAPERECORDER MAY BE NEEDED.
0000
0000          ; CURRENT EPROM VERSION WITH STARTADRESSES
0000          ;   DUMP  = $93CF (U-OUT)
0000          ;   LOAD  = $9551 (U-IN)
0000
0000          ;******************
0000          ;***  FASTDUMP  ***
0000          ;******************
0000
0000          ;VERSION 28.9.80
0000
0000 CRCK     =$EA24
0000 FROM     =$E7A3
0000 TO       =$E7A7
0000 BLANK    =$E83E
0000 ADDRS1   =$F910
0000 CRLOW    =$EA13
0000 TAPOUT   =$A435
0000 TIOSET   =$EE1C
0000 KEPR     =$E970
0000 FNAM     =$E8A2
0000 CLRCK    =$EB4D
0000 DU13     =$EA13
0000 PATC25   =$FFBC
0000 EQUAL    =$E7D8
0000 RED1     =$FE96
0000 HEX      =$EA7D
0000 NUMA     =$EA46
0000 ADDIN    =$EAAE
0000 COMIN    =$E1A1
0000 KEP      =$E7AF
0000 OUTDIS   =$EF05
0000 CKER1    =$E396
0000 BLANK2   =$E83B
0000 OUTPUT   =$E97A
0000 CKER00   =$E394
0000 CRLF     =$E9F0
0000 CLR      =$EB44
0000
0000 IFR      =$A80D
0000 PCR      =$A80C
0000 ACR      =$A80B
```

Ein Hinweis zum Listing: Das Komma wurde als '=' gedruckt. Bitte bei der indizierten Adressierung beachten!

```
0000 T1L      =$A804
0000 T1CH     =$A805
0000 T1LL     =$A806
0000 T1LH     =$A807
0000 DRB      =$A800
0000 NAME     =$A42E
0000 S1       =$A41A
0000 ADDR     =$A41C
0000 CKSUM    =$A41E
0000 DDRB     =$A802
0000 TAPIN    =$A434
0000 DIV1     =$A494
0000 PRIFLG   =$A411
0000
0000 SAVPRF   =$0115
0000
0000          *=$F0
00F0 PNTL     *=*+1
00F1 PNTH     *=*+1
00F2 NBL      *=*+1
00F3 BUFFL
00FF
00FF                *=$10A
010A          CF83  .WOR FASTD      ; U-OUT VECTOR
010C
010C                *=$108
0108          5185  .WOR FASTL      ; U-IN VECTOR
010A
010A                *=$83CF
83CF
83CF FASTD    D8    CLD
83D0          EA    NOP
83D1          20    3EE8 JSR BLANK
83D4 ASKSEC   A953  LDA #'S'        ; ASK SECURITYFACTOR
83D6          20    7AE9 JSR OUTPUT
83D9          20    D8E7 JSR EQUAL
83DC          20    96FE JSR RED1
83DF          20    7DEA JSR HEX
83E2          B0F0  BCS ASKSEC
83E4          85F9  STA SECFAC      ; STORE SECURITYFACTOR
83E6          85F8  STA SECCNT
83E8          A201  LDX #1
83EA          20    A2E8 JSR FNAM     ; GET NAME
83ED          20    4584 JSR MORQ     ; ASK IF MORE
83F0          20    5584 JSR SNSTRT   ; START TO SEND
83F3 SNDNAM   20    7384 JSR SNSYNC   ; SEND SYNCS AND NAME
83F6          A93A  LDA #':
83F8          20    3785 JSR SENDA
83FB          A000  LDY #0
83FD DNAMLP   B92EA4 LDA NAME=Y
8400          20    3785 JSR SENDA
8403          C8    INY
8404          C005  CPY #5
8406          D0F5  BNE DNAMLP
8408          A232  LDX #50         ; # OF SYNC BETW. NAME-BLOCKS
840A          C6F8  DEC SECCNT      ; SEND S TIMES
840C          D0E5  BNE SNDNAM

00F3 NBH      *=*+1
00F4 BUFFH    *=*+1
00F5 CKSFLG   *=*+1
00F6 LSTDRB   *=*+1
00F7 BYTE     *=*+1
00F8 LSLASH
00F8 SECCNT
00F8 NAMIN    *=*+1
00F9 SECFAC   *=*+1
00FA LSTBLK
00FA NXTBLK   *=*+1
00FB BITCNT   *=*+1
00FC MORFLG   *=*+1
00FD BLKCNT   *=*+1
00FE TEMPX
00FE BYTCNT   *=*+1
00FF MORTMP
00FF SLASH
```

```
840E          A296    LDX #150       ; # OF SYNC BEFORE FIRST DATA-BL.
8410          207384  JSR SNSYNC     ; ALLOWS TO RESTART TAPE
8413          A900    LDA #0         ; INITALIZE BLOCKCOUNT
8415          85FD    STA BLKCNT
8417 FASTD2   208484  JSR SNDPRT     ; SEND THIS PART
841A          A5FC    LDA MORFLG
841C          C959    CMP #'Y'       ; MORE PARTS ?
841E          D01F    BNE ENDDU
8420          2024EA  JSR CRCK
8423 FDU0     20A3E7  JSR FROM       ; ASK WHICH PART
8426          B0FB    BCS FDU0
8428          203EE8  JSR BLANK
842B          2010F9  JSR ADDRS1
842E FDU1     20A7E7  JSR TO
8431          B0FB    BCS FDU1
8433          2013EA  JSR CRLOW
8436          204584  JSR MORQ
8439          205584  JSR SNSTRT     ; RESTART TAPE
843C          4C1784  JMP FASTD2
843F
843F ENDDU    2013EA  JSR DU13       ; STOP TIMER
8442          4CA1E1  JMP COMIN      ; EXIT TO MONITOR
8445
8445 MORQ     A01C    LDY #$1C
8447          2070E9  JSR KEPR       ; OUTPUT "MORE"-MESSAGE
844A          C959    CMP #'Y
844C          F004    BEQ MORQ1
844E          C94E    CMP #'N
8450          D0F3    BNE MORQ       ; ONLY Y OR N ARE VALID
8452 MORQ1    85FC    STA MORFLG
8454          60      RTS
8455
8455 SNSTRT   AD35A4  LDA TAPOUT     ; WHICH TAPE
8458          201CEE  JSR TIOSET     ; START TAPE
845B          A9EC    LDA #$EC       ; INITALIZE VIA FOR OUTPUT
845D          8D0CA8  STA PCR
8460          A9C0    LDA #$C0
8462          8D0BA8  STA ACR
8465          A900    LDA #0
8467          8D05A8  STA T1CH       ; START TIMER BY WRITING COUNTER
846A          8D07A8  STA T1LH
846D          20BCFF  JSR PATC25
8470          A2C8    LDX #200       ; # OF SYNC FOR START
8472          60      RTS
8473
8473 SNSYNC   86FE    STX TEMPX
8475 SNSYN1   A916    LDA #$16       ; SEND 'X' SYNC
8477          203785  JSR SENDA
847A          C6FE    DEC TEMPX
847C          D0F7    BNE SNSYN1
847E          A92A    LDA #'*
8480          203785  JSR SENDA
8483          60      RTS
8484
8484 SNDPRT   AD1AA4  LDA S1         ; STARTADRESS TO POINTER
8487          85F0    STA PNTL
8489          AD1BA4  LDA S1+1
848C          85F1    STA PNTH
```

```
848E          38      SEC
848F          AD1CA4  LDA ADDR        ; CALCULATE # OF BYTES
8492          E5F0    SBC PNTL
8494          85F2    STA NBL
8496          AD1DA4  LDA ADDR+1
8499          E5F1    SBC PNTH
849B          85F3    STA NBH
849D          E6F2    INC NBL         ; ADD 1 BYTE
849F          D002    BNE SNDON
84A1          E6F3    INC NBH
84A3 SNDON    A5FC    LDA MORFLG      ; IF NBH AND NBL THEN MORTEMP IS:
84A5          A6F3    LDX NBH         ;     0       X    MORFLG(Y OR N)
84A7          F009    BEQ SNDON2      ;     1       0    MORFLG(Y OR N)
84A9          CA      DEX             ;     1      >0       'T'
84AA          D004    BNE SNDON1      ;    >1       X       'T'
84AC          A6F2    LDX NBL
84AE          F002    BEQ SNDON2
84B0 SNDON1   A954    LDA #'T
84B2 SNDON2   85FF    STA MORTMP
84B4          A5F9    LDA SECFAC
84B6          85F8    STA SECCNT      ; INITALIZE SECURITYCOUNTER
84B8 SNDON3   204DEB  JSR CLRCK       ; CLEAR CHECKSUM
84BB          A228    LDX #40         ; # OF SYNC BETW. BLOCKS
84BD          207384  JSR SNSYNC
84C0          A924    LDA #'$
84C2          8D1EA4  STA CKSUM
84C5          203785  JSR SENDA       ; SEND WITHOUT CHECKSUM
84C8          A5FD    LDA BLKCNT      ; # OF BLOCKS
84CA          202985  JSR SENDCH      ; SEND WITH CHECKSUM
84CD          A5F0    LDA PNTL        ; STARTADRESS
84CF          202985  JSR SENDCH
84D2          A5F1    LDA PNTH
84D4          202985  JSR SENDCH
84D7          A5F2    LDA NBL         ; #OF BYTES
84D9          202985  JSR SENDCH
84DC          A5F8    LDA SECCNT
84DE          202985  JSR SENDCH
84E1          A5FF    LDA MORTMP      ; Y OR N OR T
84E3          202985  JSR SENDCH
84E6          A000    LDY #0
84E8 SND1     B1F0    LDA (PNTL)=Y    ; SEND 256 BYTES
84EA          202985  JSR SENDCH
84ED          C8      INY
84EE          D0F8    BNE SND1        ; NEXT BYTE
84F0          A92F    LDA #'/'        ; SEND SLASH
84F2          202985  JSR SENDCH
84F5          AD1EA4  LDA CKSUM       ; SEND CHECKSUM
84F8          203785  JSR SENDA
84FB          AD1FA4  LDA CKSUM+1
84FE          203785  JSR SENDA
8501          C6F8    DEC SECCNT
8503          D0B3    BNE SNDON3      ; S TIMES THIS BLOCK
8505          E6FD    INC BLKCNT
8507          A5FF    LDA MORTMP
8509          C5FC    CMP MORFLG      ; MORE BLOCKS IF MORFLG = T
850B          F007    BEQ SNDEND
850D          E6F1    INC PNTH        ; NEXT PAGE
```

```
850F          C6F3    DEC NBH
8511          4CA384  JMP SNDON
8514
8514 SNDEND   A232    LDX #50         ; MOVE TAPE ON A LITTLE
8516          207384  JSR SNSYNC      ; SYNC JUST FOR TIME-FILL
8519          20BCFF  JSR PATC25
851C          8C0BA8  STY ACR         ; DISABLE PB7 FREE RUNNING
851F          AD00A8  LDA DRB         ; STOP TAPES
8522          29CF    AND #$CF
8524          8D00A8  STA DRB
8527          58      CLI
8528          60      RTS
8529
8529 SENDCH   48      PHA             ; SAVE ACCU
852A          18      CLC
852B          6D1EA4  ADC CKSUM       ; UPDATE CHECKSUM
852E          8D1EA4  STA CKSUM
8531          9003    BCC SNDCH2
8533          EE1FA4  INC CKSUM+1
8536 SNDCH2   68      PLA
8537 SENDA    A208    LDX #8          ; 8 BIT TO SEND
8539 SENDA1   0A      ASL A           ; SHIFT INTO CARRY POSITION
853A          48      PHA             ; SAVE REST OF THE BYTE
853B          A95A    LDA #90         ; TIME FOR '1'
853D          B002    BCS SENDA2
853F          A9A0    LDA #160        ; TIME FOR '0'
8541 SENDA2   8D06A8  STA T1LL        ; LOAD LATCH OF T1
8544 SENDA3   2C0DA8  BIT IFR         ; WAIT FOR TIME OUT
8547          50FB    BVC SENDA3
8549          AD04A8  LDA T1L         ; CLEAR FLAG
854C          68      PLA             ; REST OF THE BYTE
854D          CA      DEX             ; COUNT DOWN
854E          D0E9    BNE SENDA1      ; NEXT BIT
8550          60      RTS
8551
8551          ;******************
8551          ;***  FASTLOAD  ***
8551          ;******************
8551
8551          ;VERSION 28.9.80
8551
8551 FASTL    D8      CLD
8552          AD11A4  LDA PRIFLG      ; SAVE PRINTERFLAG
8555          8D1501  STA SAVPRF
8558          A200    LDX #0
855A          20A2E8  JSR FNAM        ; GET FILENAME TO LOAD
855D          8E11A4  STX PRIFLG      ; PRINTER OFF
8560          86F5    STX CKSFLG      ; CHECKSUMUPDATE OFF
8562          AD2EA4  LDA NAME
8565          C920    CMP #' '        ; IS FIRST CHAR. A BLANK ?
8567          D003    BNE BUFF1
8569          4C2587  JMP DIRE        ; YES : TAPE DIRECTORY
856C
856C BUFF1    203EE8  JSR BLANK
856F          A942    LDA #'B'        ; ASK WHICH PAGE OF RAM AS BUFFER
8571          207AE9  JSR OUTPUT
```

```
8574 BUFF2   20AEEA JSR ADDIN
8577         B0FB   BCS BUFF2
8579         C000   CPY #0          ; 'CR'=TAKE DEFAULT VALUE
857B         D00D   BNE BUFF3
857D         A98D   LDA #$8D        ;DEFAULT BUFFER PAGE
857F         8D1DA4 STA ADDR+1
8582         2046EA JSR NUMA        ; SHOW DEFAULT VALUE
8585         A900   LDA #0
8587         2046EA JSR NUMA
858A BUFF3   AD1CA4 LDA ADDR
858D         85F3   STA BUFFL
858F         AD1DA4 LDA ADDR+1
8592         85F4   STA BUFFH
8594
8594 FASTL1  209C86 JSR GETSYN      ; GET A STREAM OF SYNCS, THEN A *
8597         20E086 JSR RECV        ; GET ONE BYTE
859A         C93A   CMP #':'        ; : DENOTES A NAME BLOCK
859C         D0F6   BNE FASTL1
859E         20CF86 JSR GETNA       ; GET NAME
85A1         2044EB JSR CLR
85A4         A048   LDY #$48
85A6         20AFE7 JSR KEP         ;DISPLAY "SRCH"-MESSAGE
85A9         A200   LDX #0
85AB         A000   LDY #0
85AD CMPNA   B5F8   LDA NAMIN=X     ; COMPARE NAMES AND
85AF         2005EF JSR OUTDIS      ; DISPLAY INCOMING NAME
85B2         DD2EA4 CMP NAME=X
85B5         F001   BEQ CMP1
85B7         C8     INY             ; COUNTER FOR MISMATCHES
85B8 CMP1    E8     INX
85B9         E005   CPX #5
85BB         D0F0   BNE CMPNA
85BD         C000   CPY #0
85BF         F003   BEQ FASTL3      ; NAME O.K.
85C1         4C9485 JMP FASTL1      ; WRONG NAME
85C4 FASTL3  2044EB JSR CLR
85C7         A066   LDY #$66
85C9         2096E3 JSR CKER1       ;DISPLAY "LOAD"-MESSAGE
85CC         203EE8 JSR BLANK
85CF         A200   LDX #0
85D1         86FA   STX NXTBLK      ; FIRST LOOK FOR BLOCK 0
85D3 FASTLA  204DEB JSR CLRCK       ; CLEAR CHEKSUM
85D6         A200   LDX #0
85D8         86F5   STX CKSFLG      ; CHECKSUMUPDATE OFF
85DA         20AC86 JSR GETSY2      ; GET SYNCSTREAM= RETURN ON *
85DD         A280   LDX #$80        ; CHECKSUMUPDATE ON
85DF         86F5   STX CKSFLG
85E1         20E086 JSR RECV
85E4         C924   CMP #'$'        ;DENOTES A DATA-BLOCK
85E6         D0EB   BNE FASTLA
85E8
85E8 FASTLB  20E086 JSR RECV
85EB         85FD   STA BLKCNT      ; BLOCKCOUNT
85ED         20E086 JSR RECV
85F0         85F0   STA PNTL        ; POINTER LOW
85F2         20E086 JSR RECV
```

```
85F5          85F1    STA PNTH        ; POINTER HIGH
85F7          20E086  JSR RECV
85FA          85F2    STA NBL         ; # OF BYTES (VALID IF MORFLG<>T)
85FC          20E086  JSR RECV
85FF          85F8    STA SECCNT      ; # OF SPAREBLOCKS
8601          20E086  JSR RECV
8604          85FC    STA MORFLG      ; Y OR N OR T
8606          A000    LDY #0          ; COUNTER FOR 256 BYTES
8608          A2FF    LDX #$FF        ; FOR TIMER
860A
860A FASTL5   A908    LDA #8          ; COUNTER FOR 8 BIT
860C          85FB    STA BITCNT
860E RECB2    A5F6    LDA LSTDRB
8610 RECB1    CD00A8  CMP DRB         ; WAIT FOR CHANGE IN DRB
8613          F0FB    BEQ RECB1
8615          AD00A8  LDA DRB         ; UPDATE LASTDRB
8618          85F6    STA LSTDRB
861A          AD94A4  LDA DIV1        ; GET TIME ELAPSED
861D          8E94A4  STX DIV1        ; START TIMER FROM FF
8620          C97D    CMP #125        ; AVERAGE TIME BETWEEN 1 AND 0
8622          26F7    ROL BYTE        ; ROLL IN CARRY TO FORM A BYTE
8624          C6FB    DEC BITCNT      ; COUNT DOWN
8626          D0E6    BNE RECB2
8628          A5F7    LDA BYTE
862A          91F3    STA (BUFFL)=Y   ; STORE BYTE
862C          18      CLC
862D          6D1EA4  ADC CKSUM       ; UPDATE CHECKSUM
8630          8D1EA4  STA CKSUM
8633          9003    BCC FASTL6
8635          EE1FA4  INC CKSUM+1
8638 FASTL6   C8      INY             ; NEXT BYTE
8639          D0CF    BNE FASTL5
863B          A5FF    LDA SLASH       ; SLASH TO LASTSLASH
863D          85F8    STA LSLASH      ; (USED FOR ERRORDIAGNOSTIC)
863F          20E086  JSR RECV
8642          85FF    STA SLASH
8644          A200    LDX #0          ; CHECKSUMUPDATE OFF
8646          86F5    STX CKSFLG
8648          20E086  JSR RECV
864B          CD1EA4  CMP CKSUM       ; COMPARE CHECKSUM
864E          D008    BNE ERRCK
8650          20E086  JSR RECV
8653          CD1FA4  CMP CKSUM+1
8656          F008    BEQ FASTL9
8658 ERRCK    A945    LDA #'E'        ; ISPLAY E FOR CHECKSUM-ERROR
865A          2005EF  JSR OUTDIS
865D GETNXT   4CD385  JMP FASTLA      ; GET NEXT BLOCK
8660
8660 FASTL9   A6FA    LDX NXTBLK      ; O WE NEED THAT BLOCK ?
8662          E4FD    CPX BLKCNT
8664          F007    BEQ SHIFT1      ; YES : SHIFT BUFFER TO MEMORY
8666          10F5    BPL GETNXT      ; NO WE ALREADY HAVE IT
8668          A6F8    LDX LSLASH      ; ERROR: BLOCK MISSED
866A          4C1287  JMP ERR         ; X SHOULD CONTAIN A '/'
866D
866D SHIFT1   A000    LDY #0
866F          A5FC    LDA MORFLG
```

```
8671          C954   CMP #'T
8673          F00C   BEQ SHIFT3
8675 SHIFT2   B1F3   LDA (BUFFL)=Y  ; MOREFLAG WAS NOT T: SHIFT NBL BYTES
8677          91F0   STA (PNTL)=Y   ; AND IGNORE THE REST
8679          C8     INY
867A          C4F2   CPY NBL
867C          D0F7   BNE SHIFT2
867E          4C8886 JMP SHIFT4
8681 SHIFT3   B1F3   LDA (BUFFL)=Y  ; MOREFLAG WAS T: SHIFT ALL 256 BYTES
8           6  STA (PNTL)=Y
8685          C8     INY
8686          D0F9   BNE SHIFT3
8688 SHIFT4   E6FA   INC NXTBLK     ; INCREMENT # OF NEXT BLOCK
868A          A5FC   LDA MORFLG
868C          C94E   CMP #'N
868E          F003   BEQ FLEND      ; END ON NO MORE BLOCKS
8690          4CD385 JMP FASTLA
8693
8693 FLEND    AD1501 LDA SAVPRF     ; SET PRINTER BACK
8696          8D11A4 STA PRIFLG
8699          4CF8E3 JMP $E3F8      ; RETURN TO MONITOR
869C
869C GETSYN   A937   LDA #$37       ; INITALIZE VIA FOR INPUT
869E          8D02A8 STA DDRB
86A1          A9EE   LDA #$EE
86A3          8D0CA8 STA PCR
86A6          AD34A4 LDA TAPIN      ; WHICH TAPE ?
86A9          201CEE JSR TIOSET     ; START TAPE
86AC GETSY2   A212   LDX #18        ; # OF NEEDED SYNC
86AE          86FE   STX TEMPX      ; SAVE X
86B0 FAST1    A001   LDY #1         ; START WITH SINGLE BIT RECOVERY
86B2          20E286 JSR RECV1
86B5 FAST2    C916   CMP #$16       ; A SYNC ?
86B7          D0F7   BNE FAST1      ; TRY WITH MORE BITS
86B9          20E086 JSR RECV       ; NOW FULL BYTES
86BC          C6FE   DEC TEMPX      ; COUNT DOWN
86BE          10F5   BPL FAST2
86C0 FAST3    C92A   CMP #'*
86C2          D001   BNE FAST4
86C4          60     RTS            ; RETURN ON A *
86C5 FAST4    C916   CMP #$16       ; MUST BE A SYNC IF NOT A *
86C7          D0E3   BNE GETSY2     ; NO: LOOK AGAIN FOR SYNCS
86C9          20E086 JSR RECV       ; NEXT BYTE
86CC          4CC086 JMP FAST3
86CF
86CF GETNA    A200   LDX #0          ; GET NAME FROM TAPE
86D1 GETNA2   86FE   STX TEMPX
86D3          20E086 JSR RECV
86D6          A6FE   LDX TEMPX
86D8          95F8   STA NAMIN=X
86DA          E8     INX
86DB          E005   CPX #5
86DD          D0F2   BNE GETNA2
86DF          60     RTS
86E0
86E0 RECV     A008   LDY #8          ; SUBROUTINE FOR RECEIVING
86E2 RECV1    A2FF   LDX #$FF        ; FOR COMMENTS SEE ABOVE.
86E4 RECV2    A5F6   LDA LSTDRB
```

```
86E6 RECV3   CD00A8 CMP DRB
86E9         F0FB   BEQ RECV3
86EB         AD00A8 LDA DRB
86EE         85F6   STA LSTDRB
86F0         AD94A4 LDA DIV1
86F3         8E94A4 STX DIV1
86F6         C97D   CMP #125        ; AVERAGE TIME BETWEEN 1 AND 0
86F8         26F7   ROL BYTE
86FA         88     DEY
86FB         D0E7   BNE RECV2
86FD         C4F5   CPY CKSFLG
86FF         F00E   BEQ RECV4
8701         A5F7   LDA BYTE        ; UPDATE CHECKSUM IF FLAG IS ON
8703         18     CLC
8704         6D1EA4 ADC CKSUM
8707         8D1EA4 STA CKSUM
870A         9003   BCC RECV4
870C         EE1FA4 INC CKSUM+1
870F RECV4   A5F7   LDA BYTE        ; RETURN WITH BYTE IN ACCU
8711         60     RTS
8712
8712 ERR     2044EB JSR CLR
8715         203BE8 JSR BLANK2
8718         2094E3 JSR CKER00      ; OUTPUT "ERROR"-MESSAGE
871B         8A     TXA
871C         207AE9 JSR OUTPUT
871F         207D87 JSR DIRE3       ; PRINT
8722         4C9386 JMP FLEND
8725
8725         ; *** DIRECTORY OF TAPE AND CHECKSUMTEST ***
8725
8725 DIRE    2044EB JSR CLR
8728         A900   LDA #0
872A         85F5   STA CKSFLG
872C         207D87 JSR DIRE3       ; PRINT EMPTY LINE
872F DIRE1   204DEB JSR CLRCK
8732         209C86 JSR GETSYN
8735         20E086 JSR RECV
8738 DIRE1A  C93A   CMP #':'        ; A NAME ?
873A         D05D   BNE TEST2
873C DIRE1B  20CF86 JSR GETNA
873F         2044EB JSR CLR
8742         A92A   LDA #'*
8744         207AE9 JSR OUTPUT
8747         203BE8 JSR BLANK2
874A         A200   LDX #0
874C DIRE2   B5F8   LDA NAMIN=X
874E         207AE9 JSR OUTPUT
8751         E8     INX
8752         E005   CPX #5
8754         D0F6   BNE DIRE2
8756         A2FF   LDX #$FF
8758         86FA   STX LSTBLK
875A         AD00A8 LDA DRB         ; STOP TAPE
875D         29EF   AND #$EF
875F         8D00A8 STA DRB
```

```
8762           207D87  JSR DIRE3       ; PRINT NAME
8765           18      CLC
8766           AD00A8  LDA DRB         ; START TAPE 1 AGAIN
8769           6910    ADC #$10
876B           8D00A8  STA DRB
876E DIRE2A    204DEB  JSR CLRCK
8771           20AC86  JSR GETSY2
8774           20E086  JSR RECV
8777           C924    CMP #'$'        ; WAIT FOR FIRST DATABLOCK
8779           D0F3    BNE DIRE2A
877B           F01C    BEQ TEST2       ; BRANCH ALWAYS
877D
877D DIRE3     AD11A4  LDA PRIFLG
8780           48      PHA
8781           A980    LDA #$80
8783           8D11A4  STA PRIFLG
8786           20F0E9  JSR CRLF        ; PRINT CONTENTS OF PRINTBUFFER
8789           68      PLA
878A           8D11A4  STA PRIFLG
878D           60      RTS
878E
878E CKTEST    C924    CMP #'$'        ; AGAIN A DATA-BLOCK ?
8790           F007    BEQ TEST2
8792           C93A    CMP #':'        ; NO= IS IT A NAME ?
8794           F0A6    BEQ DIRE1B      ; YES= GET IT
8796           4C2F87  JMP DIRE1       ; NO= TRY NEXT
8799 TEST2     8D1EA4  STA CKSUM       ; READ THAT BLOCK
879C           A980    LDA #$80
879E           85F5    STA CKSFLG
87A0           20E086  JSR RECV
87A3           85FD    STA BLKCNT
87A5           20E086  JSR RECV
87A8           A900    LDA #0
87AA           85FE    STA BYTCNT
87AC           20E086  JSR RECV
87AF           20E086  JSR RECV
87B2           20E086  JSR RECV
87B5           85F8    STA SECCNT
87B7           20E086  JSR RECV
87BA TEST3     20E086  JSR RECV        ; LOOP FOR 256 BYTES
87BD           E6FE    INC BYTCNT
87BF           D0F9    BNE TEST3
87C1           20E086  JSR RECV
87C4           A900    LDA #0
87C6           85F5    STA CKSFLG
87C8 TEST6     20E086  JSR RECV
87CB           CD1EA4  CMP CKSUM
87CE           D008    BNE ERVECT
87D0           20E086  JSR RECV
87D3           CD1FA4  CMP CKSUM+1
87D6           F005    BEQ TEST7
87D8 ERVECT    A94E    LDA #'N'        ; NO CHECKSUM WAS NOT O.K.
87DA           4CDF87  JMP TEST8
87DD TEST7     A959    LDA #'Y'        ; YES CHECKSUM WAS O.K.
87DF TEST8     48      PHA
87E0           A5FD    LDA BLKCNT
87E2           C5FA    CMP LSTBLK      ; A NEW BLOCK ?
87E4           F00D    BEQ TEST8A
```

```
87E6          85FA    STA LSTBLK
87E8          2044EB  JSR CLR          ; CLEAR DISPLAY FOR NEW #
87EB          A5FD    LDA BLKCNT
87ED          2046EA  JSR NUMA         ; OUTPUT BLOCKCOUNT
87F0          203EE8  JSR BLANK
87F3 TEST8A   68      PLA              ; OUTPUT Y OR N
87F4          207AE9  JSR OUTPUT
87F7 TEST9    4C2F87  JMP DIRE1        ; NEXT BLOCK
87FA
87FA                  .END
87FA           ERRORS= 0000
```

Fortsetzung im nächsten Heft mit dem Listing für FASTLINK, der Einbindung in den Editor und in den Assembler.

¤

Produkte, Branchen- & Messebericht

Das Video-Interface für 65xx der Firma Neudecker, Berlin konnte beim Herausgeber für einige Zeit getestet werden. Merkmale: Europakarte jetzt in Industriequalität, an die Systembusse angeschlossen, kompatibel zur AIM-Expansion, wahlweise auch mit MCS 64-pol. Bus, kompatibel auch mit den dortigen Motherboards für diese Busse und die weiteren Expansionskarten 32 KRAM, ROM, I/O, Digitalinterface.

Die Taktung des Video-Interface ist bemerkenswert gelöst: Vom Quarz des Prozessors wird ein Start-Stop-Oszillator synchronisiert, der den CRT-Controller 6845 per DMA zu Zeitpunkten auf das Video-RAM zugreifen läßt, in denen der Prozessor nicht liest oder schreibt, und zwar nicht im Gegentakt mit / 1, wie in einer Applikationsschrift von Rockwell empfohlen. Im Ergebnis hat man ein absolut spratzerfreies Monitorbild mit 27 Zeilen zu 72 Zeichen. Die Zeilenzahl und die Zeichen pro Zeile sind durch Beschreiben des 6845 per Programm änderbar.

An einer mit den Prozessorbussen verbundenen Videokarte ist immer wieder die Geschwindigkeit der Ausgabe bewundernswert. Als besonders angenehm erweist sich aber das im EPROM enthaltene Betriebssystem, das nicht nur Initialisierung und Beschickung der Karte versorgt und die üblichen Cursor-Bewegungen steuert (einschl. möglichem Lichtgriffelanschluß), sondern das dem AIM auch erweiterte Funktionen im Monitor (M-Befehl in größerer Breite) und speziell in Editor und Assembler verleiht: Im normalen Betriebssystem werden Editor-Prompts wie T, B, U, D usw. nach Betätigen der Taste eigentlich störend auf den Schirm ausgegeben. Bei diesem Betriebssystem wurde das unterdrückt (Editor-Modus), stattdessen rückt der Cursor auf die jetzt aktive Zeile, ebenso auch bei F (Find) und C (Change), wenn sie schon im Bildschirm war. Andernfalls wird die aktive Zeile in die Bildmitte gestellt, so daß man vorangehende und nachfolgende Zeilen im Visier hat. Der Change-Befehl wirkt entsprechend sofort sichtbar an dieser Stelle des Bildschirmes, so daß man auch hier wieder den zusammenhängenden Text sieht. - Im Editor sind jetzt Zeilenlängen bis 80 möglich. Für den Assembler wurde ein FAST-Modus implementiert, der die Assemblierung erheblich beschleunigt.

Weitere Merkmale: Kleinschreibung durch Control-Code möglich, ebenso inverse und semigrafische Zeichen. Das Betriebssystem wird noch erweitert. Schon enthalten ist das verschiebliche Laden von Objektcode. Die Karte kann an anderen 65xx-Systemen und am 6809 betrieben werden, wenn man die EPROMs ändert. Zusammenfassung: Es handelt sich um eine leistungsfähige Video-Karte nach einem modernen Design. Als deutlich unterscheidend und als angenehm für das Arbeiten kann die erweiterte Betriebssoftware für Editor und Assembler betrachtet werden.

Beim Hofer-Drucker handelt es sich um ein elektromechanisches Interface zwischen einem Prozessor und einer Schreibmaschinentastatur. Es wird ohne Eingriffe in die Maschine oben auf die Tastatur gesetzt und betätigt per Stempeldruck die selektierten Tasten. Die Beschreibung findet sich in Heft 4/80 der FUNKSCHAU. Dieses Interface wurde in etwa eintägiger Lötarbeit beim Herausgeber assembliert. Die Dokumentation war nicht besonders hilfreich. Nach softwaremäßiger Dressur lernte der AIM 65 mit zehn Fingern blind auf einer IBM mit OCR-Kugelkopf zu schreiben und besorgte einen Teil der Assembler-Listigs in diesem Heft. Die vie-

len Zwischenräume (Spaces) ließen die entsprechende Spule sehr heiß werden und legten eine Ventilatorkühlung nahe. Das Listing für FAST und FASTLOAD (in diesem Heft) beanspruchte etwa 50 Minuten. Urteil: Preiswert (Bausatz etwa 400 DM + 13%), einfaches Prinzip (wie bei dem in den 50er Jahren mit Druckluft betriebenen Robotyper), für gelegentliche Ausdrucke in Schönschrift geeignet, nicht für den professionellen Dauereinsatz ausgelegt (Automatenbriefe), dafür auch zu langsam

In den folgenden Heften folgen Besprechungen des 12 K Extended BASIC der GWK in Herzogenrath, des 6809-Systems mit Assembler und Editor der Firma Dohmann in Gütersloh (hier schon seit einiger Zeit betrieben) und des BEM-Systems der Firma Neumüller in Taufkirchen.

Das es 65 Entwicklungs- und Testsystem der österreichischen Firma Ernst Steiner wird von der Astronic, Winzererstraße 47 d in 8 München 40 vertrieben. Es ist wohl das erste Mehrbenutzersystem mit der 6502 für bis zu 6 Terminals, mit Centronix-Interface, 2 Floppies, Betriebssystem, komfortablem Editor, Makro-Assembler und Debugger. Durch im Interrupt geschaltetes memory banking werden die Benutzer auf eigene RAM-Bereiche gelegt, sie haben gleichwohl Zugriff auf die allgemeinen Dienstleistungen des Systems.

Der ACORN ATOM ist ein britischer 6502-Personal-Computer, vertrieben von Orga Oelzner Computer-Systeme in 5 Köln, Kaiser-Wilhelm-Ring 13. Zum Preise von DM 995,; in der Grundausstattung (ohne Monitor und ohne Netzteil) wird ein Computer mit Gehäuse, ASCII-Tastatur (128 Codes), 2 KB RAM und 8 KB ROM, mit UHF-Fernsehanschluß und Cassettenrekorderanschluß geliefert, ferner mit einem 32 Bit Integer-BASIC, Mnemonic Assembler und Editor, Grafik-Prozessor. Ausbauoptionen umfassen 28 KB RAM, 4 KB mathematische Routinen im ROM (Gleitkomma, trigon. und hyperbol. Funktionen), DFÜ-Netzwerk, Parallelanschluß für Drucker, PAL-Fernsehanschluß, Farbgrafikprozessor.

Vom Hersteller wird das System für 'Hobby und Ausbildung' empfohlen. Die etwas schmalbrüstige Grundausstattung ist etwa mit dem Superboard von OHIO vergleichbar. Auch dieser Computer dürfte mit den zunehmenden Wünschen des Benutzers oft eine 'ewige Baustelle' und Kostenstelle bleiben (wie z.B. auch der AIM 65). Beachtenswert für die Preislage sind gleichwohl die grafischen Möglichkeiten, auch in der Option Farbe.

Spielprogramme gehören zum Personal Computer. Die Informationstechnik Lutz Haase, 2 Hamburg 71, Bramfelder Trift 11/IV, Tel. 040-536 04 30 implementierte eine Reihe der bekannten Spiele für den AIM 65, ebenso mathematische Routinen, z.B. für die Nullpunktbestimmung von Funktionen nach Newton. Bezug als AIM-Cassette.

Zwei eingeführte Computershops vereinigen sich zur Computershop GmbH , und zwar der Micro-Shop-Bodensee (M + R Nedela) und die Computershop GmbH (F. Monninger) in Eschborn. Das Hauptgeschäft ist in Markdorf mit Filiale in Eschborn. Man will sich auf wenige Rechnersysteme konzentrieren, für die man aus der intimen Kenntnis eine zuverlässige Beratung und Unterstützung geben kann, nämlich AIM SYKO 100, SORCERER und S-100 Systeme, wie Horizon. Der 'schnelle Belgier' (DAI) kostet jetzt nur noch DM 2.880,–. Das eingeführte Verlagsgeschäft (Zeitschrift Microcomputing/kilobaud, MICRO, Bücher) verbleibt beim MSB-Verlag M. Nedela in Markdorf.

INTERACTIVE Rockwell tut sich mit dieser speziell dem AIM gewidmeten Zeitschrift z.Zt. noch schwer. Dem Herausgeber wurde im September der hiesige Vertrieb mit der notwendigen Erstbelieferung zugesagt, die aber bisher nicht erfolgte. Auf der electronica 80 war zu erfahren, man wolle den Vertrieb von Kalifornien aus zentral organisieren wolle. Wegen Europareisen der zuständigen Herren war bis Redaktionschluß keine Klarheit zu gewinnen, wie die beim Herausgeber aufgelaufenen Abonnementsbestellungen abgewickelt werden können. Dieser Schwebezustand wird in jedem Fall bis Jahresende beseitigt, entweder mit der Lieferung von hier aus oder mit der Abwicklung durch Rockwell und Erstattung evtl. bereits geleisteter Abonnementsbezahlungen. Neue Bestellungen werden nicht mehr angenommen.

Das Buch '6502 Software Design' von Leo J. Scanlon fand mit seiner Unterweisung in Programmiertechniken und mit seinen Programmbeispielen (Maschinensprache) auch im deutschsprachigen Raum eine zustimmende und erfolgreiche Aufnahme. Ab etwa April 1981 wird eine deutsche Übersetzung zum Preise von voraussichtlich DM 38,- beim Herausgeber beziehbar sein. - Unabhängig davon ist auch ein Buch des Heruausgebers mit einem noch anderen didaktischen Zugang in Vorbereitung, in das die Erfahrungen aus zahlreichen Workshops und Beratungen einfließen.

Synertec liefert jetzt über seine Distributoren zwei neue Terminaltastaturen das KTM3 mit 40 Zeichen und das KTM3/80 mit 80 Zeichen je Zeile (389 bzw. 449 Dollar). 58 Tasten im heavy duty Plastik-Gehäuse, Darstellung von 128 ASCII-Zeichen, Video-Ausgang für 50 und 60 Hz, Baudraten 110 bis 19200. Der Zeichensatz kann durch steckbare EPROMs für den character generator auf Anwenderbedürfnisse angepaßt werden.

Ein PASCAL Computerclub formiert sich im Köln-Bonner Raum. Kontakt: Klaus Schuenemann, An der Schanz 2, 5 Köln 60, Tel. 0221- 76 01 931.

Ein Softwareverbund für Tischrechner in Kreditinstituten wird hiermit vom Herausgeber angeregt. An mehreren Stellen werden offensichtlich gleichartige Programme für das Kreditgeschäft entwickelt, meistens für PC 100 oder CBM. Die benutzten Formeln dürften gleich sein, so daß einer Weitergabe unter gewissen Bedingungen nichts im Wege stehen dürfte. Anregungen, Abgabebereitschaft und Interesse können zu einer Organisation von hier aus führen.

Die EPROM-Stecksockel für Anwenderprogramme auf der CBM-Platine sind schnell erschöpft, nachdem man verschiedene Optionen ausgenutzt hat. Ein Umstecken ist umständlich und führt zu Pin-Beschädigungen. Ein Expansionsboard für 5 bzw 15 24polige Festwertspeicher ist beziehbar bei SLS Electronic, Wagengasse 3 in 6 Frankfurt 80 und bei PHS M. Penzkofer, Teichstr. 9 in 3 Hannover 91.

Commodore wollte auf der electronica seinen neuen preiswerten (299 Dollar) VIC-20 vorführen, mußte aber wegen einer Transportbeschädigung darauf verzichten: BASIC-Rechner in der Größe eines Tastaturgehäuses, ausgestattet mit einem Commodore-eigenen Video Interfacechip für Farbausgabe, der ein vom Benutzer vorzuhaltendes Fernsehgerät/Monitor ansteuert.

In Aktion gezeigt wurde der neue Matrixdrucker 8024 mit einer Druckbreite von 132 Zeichen, 160 Zeichen pro Sekunde, mit Druckwegoptimierung in beiden Richtungen arbeitend. 96 USASCII-Zeichen können dargestellt werden, auch kundenspezifische Sonderzeichen. IEEE48-Interface.

SYCO-logic 300 ist ein AIM 65 basierender Rechner der System Kontakt GmbH in 7107 Bad Friedrichshall, Siemensstr. 5, ausgelegt für die Expansion: Im 19 "-Gehäuse mit den Hardwareoptionen für RAM/PROM oder RAM, I/O, Video, Floppy, D/A, Bus Bufferborard.

Das System Microflex 65 der Firma Rockwell wird in Kürze durch weitere Module abgerundet, die z.Zt. in Kalifornien produktionsfertig gemacht werden. Der AIM-Betreiber wird folgende Optionen für die Erweiterung seines Systems haben: 8 K stat. RAM, 32 K dyn. RAM, 16 K PROM/ROM, Floppy Disk Controller (Mitte 81), Video-Controller (Mitte 81), I/O-Modul, ACIA-Modul, IEEE 488 Controller, CPU-Karte, Adapter für 1 Karte oder Kartenkäfig mit 4 oder 8 Steckplätzen, Adapter/Buffer, Prototyping- und Extendermodul.

Ein Interface zum Anschluß von Olivetti-Schreibmaschinen an Computer wird von der Firma electronic circuits, Bleichstr. 5 in 65 Gießen 1 vorbereitet. Ein Prototyp wurde bereits auf der Hobby-Elektronik in Stuttgart gezeigt. Bei den Olivettimaschinen ET201/221 handelt es sich um Typenradmaschinen mit drei Schriftbreiten oder proportional, einem kleinen Textspeicher von 836 Zeichen, vom Mikroprozessor verwaltet und gesteuert. Die 221 hat zusätzlich ein grünes LED-Display, das den eingetasteten Text anzeigt. Natürlich sind Korrekturen in der offenen Zeile möglich. Beim Herausgeber wird eine 221 seit Mai 1980 betrieben. Sie hat die effektive Schreibleistung deutlich verbessert. - Das Interface zu DM 698,- plus Einbau macht diese Maschinen zum Terminaldrucker.

Die kleine Treiberplatine für AIM 65 , die eine Bufferung der Busse direkt am Platze der CPU vornimmt, (besprochen in Heft 15) wird bereits ausgeliefert. Nach ersten Berichten führt sie zu einem deutlich sichererem Betrieb des Computers. Die Steckerfrage konnte inzwischen auch geklärt werden:MS-8 aus der Sonderliste 4/80 von Völkner, Braunschweig. Vorgeschlagen wurden auch die Steckadapter der Serie 600 von Augat, Typen 640AG16 oder 640AG1 (ohne Gewähr). Bezug der Platine durch Bernhard Kokula, Wredestr 17 in 67 Ludwigshafen.

Neue Monitorprogramme für den AIM 65 in EPROMs werden bereits an verschiedenen Stellen implementiert, bzw. sind beabsichtigt. Man sollte davon wissen, Anforderungen erfahren und Hinweise auf die bisher aufgefallenen Fehler und Schwachstellen des bisherigen Monitorprogrammes erhalten. Die bisherigen Nennungen in dieser Zeitschrift sind sicher nicht komplett.

Michael Zimmermann, Pfungstadt

Junior Computer

1. Einleitung

Seit einigen Monaten wird ein neues auf der 6502 basierendes Selbstbausystem auf dem deutschen Markt angeboten. Propagiert wird es von der Zeitschrift ELEKTOR, die vor mehreren Jahren bereits ein SCAMP-System herausgebracht hat. Dieses wurde über die Jahre mehr und mehr ausgebaut. Beim Junior-Computer hat man einen anderen Weg beschritten. In einem Artikel in ELEKTOR von Mai 1980 wurde auf die Grundlagen eingegangen und die Einzelheiten einer Buchreihe überlassen, deren erster Band sich eingehend mit Aufbau und Einführung in den Betrieb beschäftigt.

2. Aufbau des Junior-Computer

Der Junior-Computer stellt in gewisser Weise eine modernisierte Form des KIM dar, er besteht hauptsächlich aus einer 6502-CPU, einem 2708-EPROM mit dem Monitor, einer 6532 mit I/O und Monitor-RAM und 2 Stück 2114-RAM mit 1K. Weitere Bauelemente für Decoding usw. machen das System funktionsfähig.- Dem Benutzer stehen ein Hexa-Keyboard, die KIM-bekannten Funktionstasten und 6 Siebensegment-Anzeigen für Ein- und Ausgaben zur Verfügung.

3. Beschaffung und Zusammenbau

Die Beschaffung von Platine und Bauelementen stellt kein großes Problem dar. Leidglich die Tastatur (große DIGITAST) war über mehrere Monate im Rhein/Maingebiet nicht lieferbar. Aus diesem Grund sei anderen Bauwilligen der Kauf von kompletten Bausätzen empfohlen.

Der darauf folgende Zusammenbau war problemlos. Die Platine ist solide aufgebaut, gut durchkontaktiert und macht einen professionellen Eindruck. Beide Seiten sind exakt mit Lötstoplack bedruckt, so daß auch ohne gesteigerte Sorgfalt keine Lötbrücken auftraten, auch bei durchgeschleiften Leiterbahnen nicht. Insgesamt erforderte der Aufbau ohne Hast und mit umfangreichem Durchmessen der Verbindungen rund 5 Stunden. Weitere 3 Stunden wurden für Eintasten, Disassemblieren und Austesten sowie Einbrennen des Monitors erforderlich.

4. Betrieb des Junior-Computer

Wie bereits angedeutet, baut der Junior-Computer auf dem KIM auf. Dies gilt nicht nur für die Hardware, sondern auch für die Software. Zum einen besteht der Monitor aus dem Keyboard/Display-Teil des KIM-Monitors, weiterhin wurde das SWEETS-Programm von Dan Fylstra (BYTE, Februar 1980) eingearbeitet. Hierbei werden die Grundfunktionen des KIM erweitert um die Möglichkeiten einer quasi-symbolischen Befehlseingabe.

5. Kritik

Insgesamt macht der Aufbau des Junior-Computers einen veralteten Eindruck. Statt des 2708 könnte besser der 2716 als Monitor-ROM Verwendung finden. Ebenso scheint das Konzept mit Hexa-Keyboard und Siebensegmentanzeige nicht mehr zeitgemäß. Im Layout der Platine wurde ein Fehler festgestellt: Der Pin 18 des EPROM hängt in der Luft und sollte besser an ground gelegt werden. Auch wurden für die Tastatur sehr kostspielige große Tasten gewählt, die kleineren Digitast wären genau so geeignet gewesen. Hierdurch ist der Preis,

auch für Bausätze, nicht besonders günstig. - Ein Anfänger, und an den wenden sich die Hersteller ausdrücklich, ist mit dem Original-KIM besser bedient, insbesondere weil dieser ein Kassetten-Interface hat.

Ebenso ist der 1. Band des Handbuches, der bis jetzt als einziger vorliegt, wenig geeignet, beim Anfänger tieferes Verständnis für die Datenverarbeitung zu wecken. Der mechanischen Arbeit des Eintastens wird ein breiter Raum gewidmet, vollkommen fehlt ein Assembler-Listing des Monitors, der nur als Hexa-Ausdruck vorliegt.

Gänzlich allein gelassen ist man jedoch bei Fehlern. Hier werden nur wenig Hinweise auf das Erkennen und das Beheben gegeben. - Vom Verfasser wurde der Junior-Computer durch Drahtbrücken auf das 2716 EPROM und damit auf eine einzige 5-Volt-Spannung umgestellt und dient hier für Experimente mit unterschiedlichen Monitoren. Hierbei ist er dem KIM durch leichte Austauschbarkeit der EPROMs überlegen.

Tastentest für CBM 3032

Das nachfolgende Programm dient dem Testen der großen Tastatur des CBM. Es ist zugleich ein Beispiel für den Gebrauch komprimierter Tabellen. Per BASIC wird die Tastatur, so gut es geht, auf dem Bildschirm abgebildet. Das Maschinenprogramm invertiert bei Tastendruck das Zeichen auf dem Bildschirm. Der Tastaturcode wird der Zeropage-Adresse 166 (hex A6) entnommen. Mit Hilfe der Tabelle wird daraus die Adresse des Zeichens auf dem Bildschirm bestimmt. Das Programm wird mit SHIFT+STOP beendet. Es liegt im 2. Kassettenbuffer und kann zusammen mit BASIC geladen werden.

```
10 PRINT"■■■■":POKE32933,34
20 PRINT"┌─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┐  ┌─┬─┬─┐
30 PRINT"│@│!│"│#│$│%│&│'│(│)│←│(│)│ │  │■│■│■│
40 PRINT"├─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┤  └─┴─┴─┘
50 PRINT"│■│ R │■│Q│W│E│R│T│Y│U│I│O│P│↑│<│>│7│8│9│/│
60 PRINT"├─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┤
70 PRINT"│ │ │A│S│D│F│G│H│J│K│L│:│;│■│C■│■│ M │■│4│5│6│*│
80 PRINT"├─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┤
90 PRINT"│■│ │■│Z│X│C│V│B│N│M│,│.│/│?│■│SHIFT■│1│2│3│+│
100 PRINT"└─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┴─┬─┘
110 PRINT"       │■│   S P A C E   │■│       │0│.│-│=│
120 PRINT"       └─┴─┴─┴─┴─┴─┴─┴─┴─┘       └─┴─┴─┴─┘
200 SYS947
```

```
MASCHINENPROGRAMM FUER TASTENTEST

038A  A2 03     LDX #03
038C  20 91 03  JSR 0391
038F  A2 0B     LDX #0B
0391  A9 40     LDA #40
0393  85 5F     STA 5F
0395  BD 39 03  LDA 0339,X
0398  0A        ASL A
0399  26 5F     ROL 5F
039B  69 A1     ADC #A1
039D  85 5E     STA 5E
039F  90 02     BCC 03A3

03A1  E6 5F     INC 5F
03A3  A0 00     LDY #00
03A5  B1 5E     LDA (5E),Y
03A7  C9 5D     CMP #5D
03A9  F0 07     BEQ 03B2
03AB  49 80     EOR #80
03AD  91 5E     STA (5E),Y
03AF  C8        INY
03B0  D0 F3     BNE 03A5
03B2  60        RTS
```

```
03B3  A6 A6     LDX A6
03B5  30 0A     BMI 03C1
03B7  20 91 03  JSR 0391
03BA  E4 A6     CPX A6
03BC  F0 FC     BEQ 03BA
03BE  20 A3 03  JSR 03A3
03C1  A5 98     LDA 98
03C3  F0 EE     BEQ 03B3
03C5  20 8A 03  JSR 038A
03C8  A5 98     LDA 98
03CA  D0 FC     BNE 03C8
03CC  20 8A 03  JSR 038A
03CF  20 01 F3  JSR F301
03D2  D0 DF     BNE 03B3
03D4  60        RTS
```

```
.
.  TABELLE
.
.:  033A B2 B0 78 5C 35 A3 0C 28
.:  0342 B1 AF 84 36 FF 0D 00 FF
.:  034A 8A 88 FF 83 81 7F 7D 7B
.:  0352 89 87 5D 82 80 7E 7C 7A
.:  035A 62 60 FF 5B 59 57 55 53
.:  0362 61 5F FF 5A 58 56 54 52
.:  036A 3A 38 FF 33 31 2F 2D 2B
.:  0372 39 37 34 32 30 2E 2C 2A
.:  037A 12 10 FF 0A 08 06 04 02
.:  0382 11 0F 0B 09 07 05 03 01
```

¤

Buchbesprechungen

Applikationen zum Personal-Computer PC 100, Ausgabe 1980/81; Siemens Aktiengesellschaft, 71 Seiten, Bestell-Nr. B/2367.

Auch mit dieser 'Technischen Dokumentation' wird der PC 100 transparenter gemacht. Folgende Themen sind dargestellt: Serielle Schnittstelle V24, Druckerbetrieb mit Parallelschnittstelle, Kassettenrekorderanschluß, Hilfsprogramm zum Einlesen und Abspeichern von Daten mit dem BASIC-Interpreter, INPUT-Funktionen als Unterprogramme, EPROM-Programmierer PC 100, BASIC-Programme auf ROM oder EPROM, Einlesen von 4 1/2 Stellen BCD-Daten mit Auswertung, Terminal PC 100, Hinweise, Erklärungen, Tabellen.

Ersichtlich handelt es sich um häufig für den Computer benötigte Dienste, deren Darstellung dankenswert ist. Dem Anwender wird die Inbetriebnahme Schritt für Schritt nahegebracht. Bei künftigen Ergänzungen würde man allerdings wünschen, daß statt BASIC mehr maschinensprachliche Programmierung verwendet wird, denn sie ist in steuernden Dingen transparenter.

Assembler-Handbuch Personal-Computer PC 100, Ausgabe 1980/81, Siemens Aktiengesellschaft, 121 Seiten, Bestell-Nr. B/2327.

Im handlichen Buchformat A5 legt das Haus Siemens jetzt ein Handbuch vor, das der Programmierung in Assemblersprache dient, das eine Tabelle aller Maschinenbefehle und ihrer Adressierungsarten sowie die Erklärung der Befehle des Monitor-Programmes enthält. Man könnte es als einen konzentrierten Querschnitt durch das Programmierhandbuch für 6502 und das Anwenderhandbuch für AIM 65 bezeichnen. Beim näheren Hinsehen findet man speziell für den Assembler zahlreiche neu herausgearbeitete Schritt-für-Schritt-Unterweisungen und sogar die Implementierung neuer Assembler-Anweisungen. Auch für den Monitor sind in übersichtlicher Darstellung praktische Hinweise enthalten. - Als sorgfältig aufgemachtes Handbuch für den Arbeitstisch - es erhebt keinen Anspruch, eine Unterweisung für das Programmieren zu sein - darf es den Systembetreibern empfohlen werden.

Lexikon der modernen Elektronik, bearbeitet von Reihold Falkner. Verlag Markt & Technik, München 1980, ISBN 3-922 120-05-9, 232 Seiten DM 48,-.

In der Beschreibung heißt es: Aus dem Englischen übersetzt und ausführlich erklärt. - Dieses Lexikon wurde von einem Team von Elektronik-Spezialisten zusammengestellt und enthält mehr als 2000 der wichtigsten Fachbegriffe aus den Gebieten Allgemeine Elektronik, Mikroelektronik, Mikrocomputertechnik und Mikrocomputer-Software. Dem englischen Ausdruck ist jeweils die deutsche Übersetzung zugeordnet. Anschließend wird jeder Begriff exakt und ausführlich erklärt.

Gemäß Beschreibung ist nur eine Teilmenge der Informationen dem Gebiet der Mikrocomputer gewidmet. Für diesen Bereich ist zu fragen, wie klar das Lexikon die Fachbegriffe darlegt. - Die meisten Benutzer dürften schon nach der Prüfung einfacher geläufiger Begriffe enttäuscht sein: Port, Vektor, Adresse, Adressierungsart, Datenbus, Interface, Hamming Code, Timer, Stromschleife/TTY usw.. Es bleibt daher sorgfältig abzuwägen, ob die nicht geringe Ausgabe speziell für Mikroprozessoren lohnt.

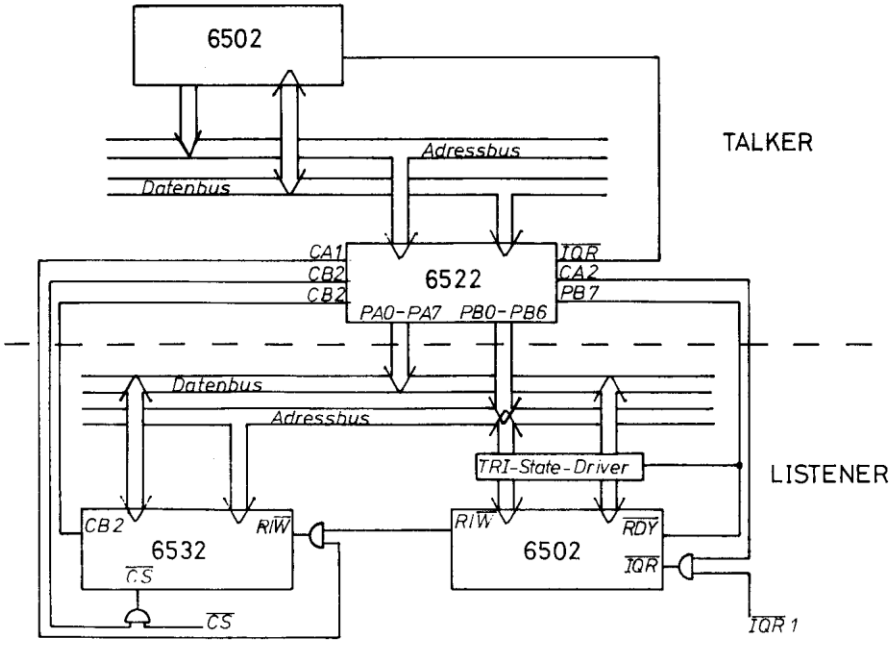
¤

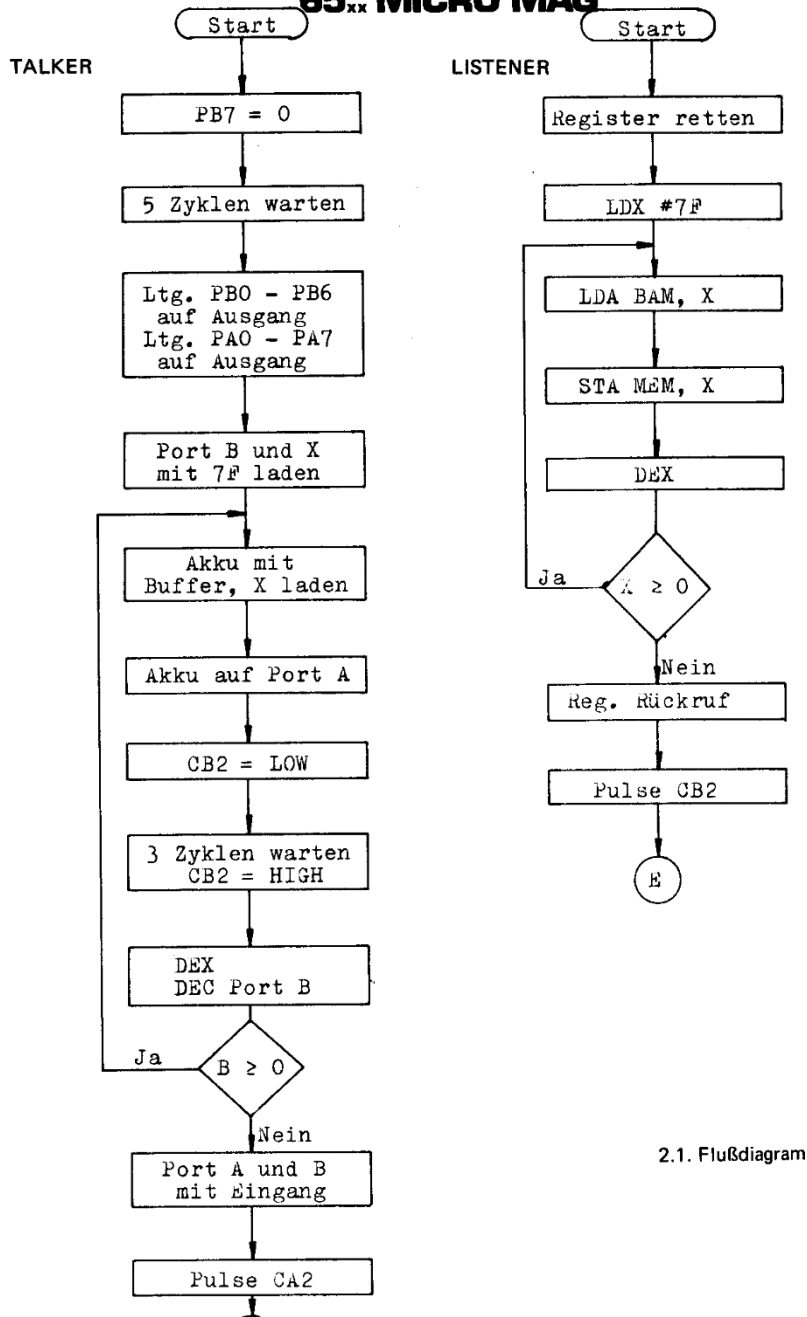
Dipl.-Ing. U. Kornnagel, Ing. grad. G. Krohn und Ing. grad. P. Bach

Datenaustausch zwischen zwei Mikroprozessorsystemen (2)

2. Datenaustausch mit Buszugriff

In den nachfolgenden Darstellungen wird eine Beschaltung behandelt, bei der die an den VIA-Ports des Talkers bereitgestellten Daten direkt in das RAM eines 6532 RIOT-Bausteines (RAM, I/O, Timer) eingeschrieben werden können:





2.1. Flußdiagramm

2.2. Das TALKER-Verhalten

Der TALKER setzt die Ausgabeleitung 7 des Port B auf LOW. Damit hält er den Prozessor des LISTENERs an und signalisiert, daß Daten anstehen. Gleichzeitig schneidet er über den TRI-State Driver die CPU vom Daten- und Adreßbus ab. Nunmehr hat er freien Zugriff auf den hierfür vorgesehenen Speicherbereich des Listeners (Backup Memory). Als nächstes setzt der Talker seine Datenrichtungsregister auf Ausgabe, außerdem wird ein Indexregister (Zähler) und Port B (0-6) als Adreßgenerator mit 127 (hex 7F) geladen. Nun kann der Talker in einer Schleife 127 Bytes in das Backup Memory schreiben. An Port A wird eine Information (1 Byte) angelegt und auf der Leitung CB2 ein negativer Impuls erzeugt, damit das RAM im 6532 einen Schreibbefehl erkennt. Um sicherzugehen, daß die Daten übernommen wurden geht CB2 erst nach 3 Taktzyklen wieder auf HIGH. - Die Adresse und der Zähler werden solange um 1 erniedrigt, bis ein negativer Überlauf errreicht ist. Die Ausgabe wird dadurch beendet, indem die Ports auf Eingabe geschaltet werden. Durch einen Impuls auf die Leitung CA2 löst der Talker beim Listener einen Interrupt aus. Damit wird angezeigt, daß das Backup Memory gefüllt ist.

2.3 Das LISTENER-Verhalten

Durch die Umschaltung der Ausgabeleitungen des TALKERS auf Eingabe erhält die CPU des LISTENERs die Kontrolle über ihren Adreß- und Datenbus zurück. Durch den von CA2 ausgelösten Interrupt erkennt die CPU, daß gültige Daten in das Backup Memory geladen wurden und verzweigt zur Verarbeitungsroutine. Hierbei wird der Registerinhalt gerettet und die Daten des Backup Memory in einen vom Anwender zu bestimmenden RAM-Bereich geladen. Wenn der ursprüngliche Registerinhalt zurückgerufen ist, wird über CB2 ein Impuls zum Talker gesendet, der die Verarbeitung der empfangenen Daten anzeigt.

2.4. Programm

```
;TALKER PB7 AUF Ø
LDA #$80
STA DDRB
STA DRB
NOP      WARTE
NOP      5 ZYKLEN
NOP
NOP
NOP
LDA #$FF LEITUNGEN
STA DDRA PAØ-PA7 UND PBØ-PB7
STA DDRB AUF AUSGABE
LDX #$7F PORTB UND X
;MIT 7F LADEN
;AUSGABESCHLEIFE
LOOP LDA BUFFER,X
STA DRA
;CB2 LOW
LDA PCR
AND #$DF
STA PCR
;WARTE 3 ZYKLEN
NOP
NOP
NOP
;CB2 HIGH
LDA PCR
ORA #$40
STA PCR
;DEC PORT B UND X
DEC DRB
DEX
BPL LOOP

;PORT A UND B AUF EINGANG
LDA #$0
STA DDRA
STA DDRB
;PULSE CA2
LDA PCR
AND #$FD
STA PCR
ORA #$02
STA PCR
RTS

;LISTENER
 ;REGISTER RETTEN
 PHA
 TXA
 PHA
TYA
 PHA
;X MIT 7F LADEN
LDX #$7F
;EINLESESCHLEIFE
LOOP LDA BAM,X
STA MEM,X
DEX
BPL LOOP
;REGISTERRUECKRUF
PLA
TAY
PLA
TAX
;PULSE CB2
LDA PCR
AND #$DF
STA PCR
ORA #$40
STA PCR
PLA
RTI
```

Wird fortgesetzt. ¤

VIDEO - KARTE

1 Bildschirmseite	80 Zeichen x 24 Zeilen
	oder
2 Bildschirmseiten	64 Zeichen x 16 Zeilen
	oder
	beliebiges Format
2 k RAM	Bildwiederholungsspeicher
	vom Prozessor freier Zugriff, sehr schnell
2 k RAM	Zeichengenerator
8 x 8	Punkt-Matrix Zeichengenerator
16 x 8	max. darstellbares Raster (Unterlängen)
	Beispiel: m^2 (Hochzahlen) $1A_{hex}$ (Indices)
12 x 7	Punkte Zeichenformat eingestellt bei Auslieferung
128 ASCII	Zeichensatz erweiterbar auf 256 Zeichen on board
	frei programmierbar
	dadurch
Grafik	bzw. Semi-Grafik möglich
Adresse	in 4 k Schritten einstellbar
	Bustreiber, VG64 Steckerleiste
Monitor	in 2 k EPROM auf der Karte inkl. Standard-Zeichensatz
Cursor	frei programmierbar
Control-Codes	werden vom Steuerprogramm ausgeführt
Invers	durch Control-Code
Halbe Helligkeit	statt Invers möglich
Lichtgriffel	anschließbar

Preis: DM 660,-- zzgl. MWSt

Weiterhin lieferbar:
CPU-Karte NICO 65 mit 6502 CPU
CPU-Karte NICO 69 mit 6809 CPU
Combo-Karte mit 8 k RAM und 2 VIAs
Buskarte
EPROM-Programmiergerät

ELEKTRONISCHE BAUGRUPPEN
MICROPROZESSOR-TECHNIK
Dohmann

Im Wiehagen 15 - Tel. (05241) 67480
4830 Gütersloh 1

Systemerweiterung für AIM 65 / PC 100 auf Europakarten

Einheitliches Format 100x160 mm, einheitlich nur +5 Volt. AIM-Expansion-Bus kompatibel, "S44-Bus", AIM-Software kompatibel, alle Karten auch mit 64-poligen a/c DIN-Stecker (MCS-Bus) lieferbar. Adreßdekodierung auf jeder Karte, anschlußfertig und ausführlich getestet, 1 Jahr Garantie. - Für den PC 100 gibt es Sonderausführungen, die mit der im Gerät vorhandenen Stromversorgung auskommen.

32 KB RAM-Modul - die meistgekaufte AIM-Speichererweiterung!
792,- + MWSt = 894,96 DM

teilbestückt mit 16 K: 526,- + MWSt = 594,38 DM

VIDEO-Interface 615,- + MWSt = 694,95 DM
mit erweitertem Betriebssystem für Editor, Assembler, Monitor.

Analog I/O 8 Bit, 80 kHz max. Abtastrate, mit Aliasingfiltern und sample and hold, I/O unabhängig voneinander.
370,- + MWSt = 418,10 DM

EPROM-Karte 32 KB für 8 Stück 2716 und 2532 oder 4+4 St. gemischt
ohne EPROMs 180,- + MWSt = 203,40 DM

BUS EXPANSION zusätzlich gepuffert, 6 Steckplätze, erweiterbar, lieferbar für 44- und 64-polige Karten
240,- + MWSt = 271,20 DM

ICs (Preise incl. MWSt), jeweils das Spitzenfabrikat (F, NEC, Motorola, TI etc.):

2114L/450 = DM 10,-	2114 CMOS = DM 24,-	4116/200 = DM 15,-
2708L/450 = DM 17,-	2716 = DM 28,-	2532 = DM 58,-
6502 = DM 28,-	6522 = DM 28,-	6532 = DM 32,-
6821 = DM 15,-	6845 = DM 74,-	NB8866 = DM 120,-

Centronix-Matrixdrucker 737 7x9 Matrix, 3 Schrifttypen, auch Proportionalschrift, 3 Papierzuführungen, Mikroprozessorelektronik, Randausgleich und mehr Eigenschaften (Prospekt anfordern!).
Mit Centronix-Interface und AIM-Interfacesoftware DM 2.350,-

DIPL.-ING. HORST NEUDECKER

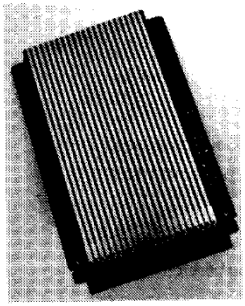
INGENIEURBÜRO FÜR MESSTECHNIK

MEHRINGPLATZ 13
1000 BERLIN 61

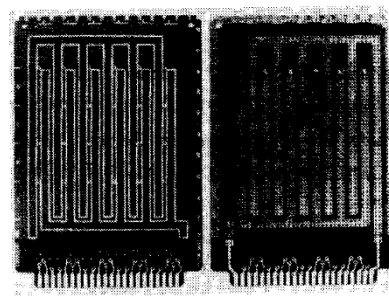
TEL.: 030- 614 89 00
030- 251 20 00

Verkauf von Leiterplatten, Steckadapter, Relaisplatinen

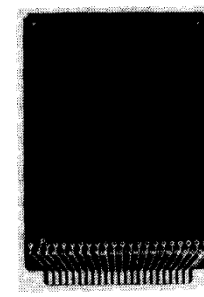
1



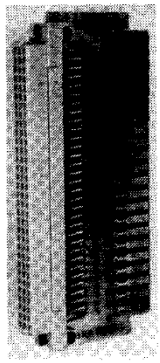
2



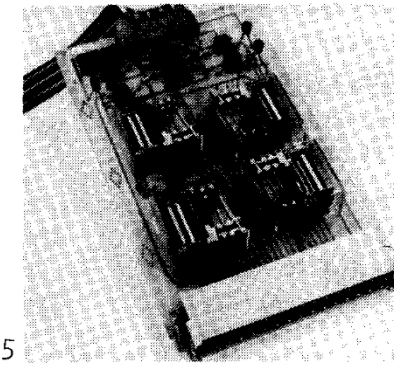
3

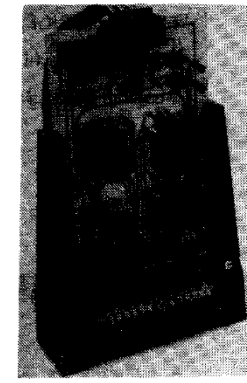


4



5





	p.St.	bei Abnahme von 3-5	von 6-10
1. 3690 Card Extender	DM 67,95	DM 57,75	DM 50,95
2. 3682-2 DIP Plugboard	DM 34,05	DM 28,95	DM 25,55
3. 3662 DIP Plugboard	DM 31,10	DM 26,45	DM 23,35
4. C 991 Steckadapter	DM 59,50	DM 50,55	DM 44,65
5. E 991 Relaisplatine	DM 210,60	DM 179,05	DM 157,95
6. E 941 dito m. Halter	DM 261,80	DM 221,85	DM 201,35

Alle Preise zuzüglich MWSt und Versandspesen. Lieferung ab Lager Köln, Zwischenverkauf vorbehalten. Weitere techn. Beschreibung auf Anfrage.

STECKER
INGENIEURBÜRO

5000 KÖLN 60 (Nippes)
Postfach 60 07 66
Delmenhorster Str. 20
Telefon (02 21) 74 51 12

12 KByte EXTENDED BASIC

für

AIM 65 / PC 100

Sehr umfangreiches und komfortables BASIC,
u.a. mit folgenden zusätzlichen Befehlen:

INDEVICE	OUTDEVICE	RENUMBER
EDIT	QUIT	FORMSTR$
HEXSTR$	BINSTR$	INSTRING
INPUT LINE	AUTO	HELP
ON ERROR GO TO	XOR	TRACE
DPEEK (16 Bit)	DPOKE (16 Bit)	

Preis: DM 445,-- + MWSt + Datenträger

Erhältlich auf Cassette, Diskette, EPROM.

GWK Gesellschaft für Technische Elektronik mbH.

D-5120 Herzogenrath

Asternstraße 2

Tel.: 026 06 - 623 94

65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

Herausgeber:
Dipl.-Volkswirt Roland Löhr
Hansdorfer Straße 4
D–2070 Ahrensburg
Tel.: 04 102 - 55 816

65xx MICRO MAG erscheint zweimonatlich. Beiträge, die nicht besonders gekennzeichnet sind, stammen vom Herausgeber. - Offsetdruck: Druckartist Gerhard M. Meier, Hamburg 70

Bezugsbedingungen: Abonnement ab laufender Ausgabe für 6 Hefte DM 45,-- (Inlandsendpreis). Ausland/Foreign via surface mail DM 50,--, USA air $ 30.

Die Hefte 1-6 sind nur noch als Buch lieferbar, siehe Anzeige unten. Die Hefte 7 bis 15 können einzeln oder komplett nachgeliefert werden, und zwar zum Preise von DM 7,80/Stück.

Private Besteller werden um Überweisung oder Scheck zusammen mit der Bestellung gebeten. Richten Sie bitte Ihre Überweisungen auf das Konto Roland Löhr, Nr. 20/01121 bei der Vereins- und Westbank in Ahrensburg, BLZ 200 300 00. Eine Zuschreibung ist auch über das Postscheckkonto 2244-207 PSchA Hamburg der Vereinsbank möglich, wenn o.a. Name und die Kontonummer auf dem Empfängerabschnitt vermerkt sind.

Leser-Service des Herausgebers

Thermopapier

für AIM 65/PC 100 in kontrastreicher Spitzenqualität
Packung mit 8 Großrollen á 65 m = 520 m, preiswert DM 50,85

Printerplatte für AIM 65/PC 100 m. Einbauanleitung DM 21,--

Disketten 5,25", softsektoriert, einseitig, single density (z.B. für CBM 3040), vom Hersteller 100%-ig geprüft. 10 St. in Plastik-Archiv-Aufstellbox DM 89,--

Anwenderhandbuch für AIM 65, deutsch

Original Rockwell-Handbuch DM 32,10

Das Buch 1-6 des 65xx MICRO MAG

ca. 230 Seiten, Sammelband der Hefte 1-6 dieser Zeitschr. gebunden, o. Anzeigen. Wertvolles Arbeitsbuch DM 26,--

Microprocessor Systems Engineering

von Camp, Smay, Triska, englisch. Lehrbuch für 65xx/AIM 65 DM 86,--

6502 Software Design

v. L.J. Scanlon
emfehlenswert (s.S. 59), engl. 270 S. DM 29,--

Vorstehende Preise sind Endpreise, einschl. Versand. NN + DM 1,50

Programmierworkshops 6502 & 6809

in Ahrensburg bei Hamburg. 6502 Assembler- und Interfaceprogrammierung für Fortgeschrittene am 23. und 24.1.1981. 6502 Programmierseminar für Anfänger, Befehlssatz, Programmiertechniken, Interfacebausteine am 30. und 31.1.1981. 6809 Workshop: Befehlssatz, Assembler- und Interfaceprogrammierung am 13.und 14.2.1981. Sonderprospekt beim Herausgeber. Schulungen in Firmen auf Anfrage.