65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DM 7,- Nr. 7 Juni 1979

Zum Start ins zweite Jahr

Im Juni 1978 erschien das 65xx MICRO MAG als wohl erste deutsche Fachzeitschrift für Mikroprozessoren. Der verlegerische und redaktionelle Grundgedanke, für eine der sehr wichtigen Prozessorfamilien möglichst viel und intensive Information zu bringen, mit Schwerpunkt bei der Software für die Systementwicklung, fand bei einer stetig wachsenden Leserschaft Zustimmung. Es ist eine Leserschaft aus vorwiegend beruflich oder wissenschaftlich befaßten Fachleuten geworden, der Untertitel 'Hobby' will nicht mehr so recht passen.

Das erste Jahr brachte den Leser viele grundlegende Darstellungen und Anregungen für die eigene Entwicklungsarbeit und begründete einen fruchtbaren Gedankenaustausch, der das 'Salz' der von hier aus für die Leser ausgeübten Dienstleistung ist und den es auch künftig sehr zu fördern gilt.

Dieses Heft bringt wiederum wichtige Darstellungen. Das leistungsfähige VIA 6522 wird mit seinem hardwaremäßigen Vermögen und mit zahlreichen Programmierbeispielen intensiv besprochen. Mit dieser Ausarbeitung und Übersicht soll der Leser auf die Prinzipien hingewiesen werden, wie die hochentwickelten programmierbaren Peripheriebausteine zu nutzen sind.

Nach den Beobachtungen konzentriert sich das Leserinteresse weiterhin sehr auf den AIM 65 und auch auf den PET. Für AIM wird daher aktuell die Handhabung des Assemblers besprochen, die im Anwenderhandbuch leider etwas un-

Inhaltsverzeichnis

glücklich dargestellt ist. Und es wird natürlich auch das inzwischen recht zügig lieferbare AIM-BASIC vorgestellt. Ein interaktives Systemprogramm zur Verschiebung und Umrechnung von Anwenderprogrammen schließt sich an.

Für PET wurde der in Heft 6 abgedruckte 6502-Assembler verbessert (weitere Vorschläge sind sehr willkommen). Ein vielseitiger Video-Driver als Maschinenprogramm von weniger als 200 Bytes tritt hinzu.

Viele Leser betreiben Prozessoren verschiedener Familien nebeneinander. Es wird ein Weg aufgezeichnet, wie der Datenaustausch zwischen ihnen via Tonbandcassette stattfinden kann. Diesem Datenaustausch sollte künftig mehr Aufmerksamkeit zugewandt werden, zumal sicher viele KIM-1 und andere Verwendung als front-end-processsor finden sollen. Der ebenfalls abgedruckte Rauschgenerator wird sicher besonderes Interesse für die Generierung von Zufallszahlen finden.

Wegen des Umfanges der genannten Artikel können die Fortsetzungsserien 'Advanced Subroutine Package' und 'Leitfaden für die Programmierung' erst im nächsten Heft weitergeführt werden, das wegen der Sommerpause gegen Ende August erscheint.

Wie sich der Einsatz neuer Satztechniken bewährt, bleibt zu beobachten. Es gilt weiterhin der Primat von Information und Aktualität unter Verzicht auf unnötige Aufmachung. Und schließlich müssen die umfangreichen Programmlistungen weiterhin sorgfältig manuell erstellt werden.

* *

Wie inzwischen bekannt sein dürfte, bietet auch das Haus Siemens einen 6502-Mikrocomputer an, den PC100. Im Kern handelt es sich um einen AIM 65, jedoch betriebsbereit ausgerüstet mit Netzteil und einem ansprechenden Gehäuse. Weitere Ausbauplatinen sind beabsichtigt. Ob dieser 'Persönliche Computer' vorwiegend als BASIC-Rechner Aufnahme findet, wie man auf der Hannover-Messe als Erwartung hörte, bleibt abzuwarten. Nach aller Erfahrung wird sein Vermögen beim Anwender den Wunsch nach Programmierung und Steuerung auch in Maschinensprache wecken.

Das Angebot an Ausbauplatinen und 65xx-basierenden Systemen wird größer. Leider sind noch nicht alle zugesagten Datenblätter und Beschreibungen hier eingetroffen, in Kürze wird der Leser sicher Anzeigen weiterer Firmen finden. In Hannover stellte die GWK ihr Euroboard-System im Einbaugehäuse erfolgreich vor.

Eine bemerkenswert ausgestattete CPU-Karte, die inzwischen in Serie ging, konnte der Herausgeber vor einiger Zeit bei der Firma I. Dohmann in Gütersloh im praktischen Einsatz sehen. Sie ist als Anwenderkarte im Europaformat konzipiert und enthält neben der CPU und 2 kRAM Stecksockel für 8 k Anwenderprogramm, 2 VIAs und bereits Treiber für alle Busse. Diese NICO 65-Karte wird ergänzt durch eine RAM/VIA-Karte (2 VIAs als Option) und eine TV-Karte. Für viele Kleinserien sicher eine preiswerte Lösung für die Hardware.

****

AIM 65 - Seminare ab September 1979

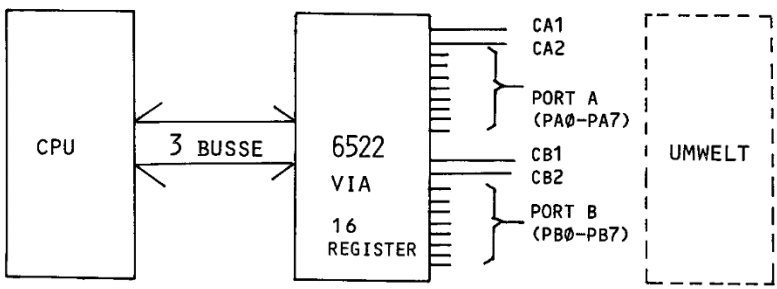
In verschiedenen Ballungsgebieten werden ab September 3-tägige Intensivschulungen am AIM 65 abgehalten. Dauer jeweils von Freitag bis Sonntag mittags. Jeder Lehrgangsteilnehmer erhält mit den besonders ausgearbeiteten Lehrgangsmaterial einen AIM 65 für die praktische Arbeit. Mehrere erfahrene Dozenten unterrichten nach straffem Plan in Hardware und vor allem in der Programmierung. Zielgruppe: Leitende und qualifizierte Mitarbeiter. Informationen durch den Herausgeber, Tel. 04102 - 55 816.

DAS VIA 6522

E: The Versatile Interface Adapter, or as many people say, the Very Versatile Interface Adapter, is described in many detailed applicational examples, scoping interfacing of I/O-ports, detecting active transitions on the inputting peripherial control lines, handshaking for I/O-control, the versatile timers which count and generate square waves and control serial I/O. A last chapter covers interrupt control.

In den neueren Computern, AIM 65, SYM-1, PET u.a., ist das VIA z.T. mehrfach als Interfacebaustein enthalten. In einigen Fällen betreibt es dabei die Hardware des Systems, in anderen steht es dem Benutzer für seine Applikationen zur Verfügung. Unabhängig von den Datenblättern wollen wir daher einmal seine Funktionsweise erläutern und verschiedene Einsatzmöglichkeiten darstellen.

In der blockmäßigen Vereinfachung gemäß folgendem Bild können wir das VIA als einen mit der CPU über Adreß-, Daten- und Steuerbus verbundenen Interfacebaustein mit 16 Registeradressen bezeichnen, der zur Verbindung mit der Umwelt 2 Ports mit je 8 E/A-Pins und 4 Pins für Steuerfunktionen trägt. Insgesamt stehen damit 20 E/A-Leitungen zur Verfügung.



Das VIA weist viele Ähnlichkeiten mit den anderen, z.T. bekannteren Interfacebausteinen der Prozessorfamilie auf, z.B. dem 6520 PIA und dem 6530 RRIOT im KIM. Das VIA vermag aber wesentlich mehr als die eben genannten:

1. Verbindung zur CPU und Adressierung

Im Konzept der 65er Familie werden die Register der Interfacebausteine von der CPU wie ganz normale Speicherzellen gelesen und beschrieben. Daher der volle Anschluß an den Datenbus und an die Steuerleitungen R/W und Ø 2. Um Pins zu sparen, werden nicht alle 16 Adreßleitungen auf den Chip gelegt, sondern nur 4 als Register Select (RSO-RS3). Das reicht zur Ansprache der 16 Maschinenregister aus. Und über eine externe Dekodierlogik wird aus den Signalen der höheren Adreßbusleitungen ein Chip-Select gebildet (CS1, CS2).

Je nach dem bei der Dekodierung betriebenen Hardwareaufwand mag sich dabei für mehrere Interfacebausteine eine lückenlose Aneinanderreihung von Registeradressen ergeben oder auch nicht. Im letzteren Fall mag ein Baustein wegen zu grober Dekodierung die Adressen einer ganzen Page oder gar eines ganzen k Byte belegen. Für eine spätere Expansion des Systems im vorhandenen Adressenraum kann das hinderlich sein.

Zur Verbindung mit dem Steuerbus gehört auch die Interrupt-Leitung IRQ. Damit ist nicht gesagt, daß das VIA im Interrupt betrieben wird. Vielmehr: Es darf einen Interrupt auslösen, wenn der Programmierer das für gewisse Ereignisse vorgesehen hat.

2. Funktionskontrolle durch 16 Register

Interfacebausteine sind keine Digitalbausteine mit ziemlich eng festgelegten Funktionen, sie sind in ihrem Einsatz vielmehr wandelbar. Der Benutzer 'initialisiert' sie durch Einschreiben in die Register. Dadurch werden auf dem Baustein selbst die benötigten digitalen Funktionen in der benötigten Art durchgeschaltet. Man könnte also sagen, es sind programmierbare Digitalbausteine. Und mit der Zahl ihrer jeweiligen Register steigt auch ihre Vielseitigkeit. Es sei erwähnt, daß der CRT-Controller R6545 (ab Juli 1979 in Mustern) immerhin schon 23 Register trägt.

Auf die Register des VIA kommen wir noch im einzelnen zu sprechen. Wir gliedern sie zunächst schematisch wie folgt:

Zwei Datenrichtungsregister, DDRA und DDRB, die die Funktionen der Ports bitweise (pinweise) für Eingabe oder Ausgabe festlegen.

Zwei Ports mit Ausgaberegister ORA, bzw. ORB, Leseregister IRA (IRB) und einschaltbarem Lese-Latch.

Zwei 16 Bit breite Timer, die zahlreiche Funktionen ausüben können, wie Impulsweitenmessung, Ereigniszählung, Erzeugung von einmaligen oder stetigen Rechteckimpulsen an PB7 und Kontrolle seriellen Datenverkehrs.

Schieberegister SR für die serielle Ein- und Ausgabe an Pin CB2.

Hilfsregister ACR, das die Funktionen der Timer, der Pins PB7 und PB6 und die Vorspeicherung einkommender Signale an den Latches der Ports A und B reguliert.

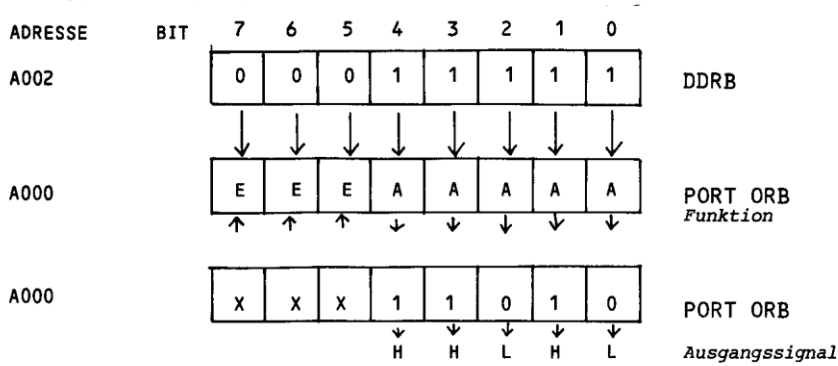
Steuerregister PCR zur Funktionskontrolle der Pins CA1, CA2, CB1 und CB2 für die Reaktion auf Impulsflanken und für den Handshake-Verkehr mit Peripherieeinheiten.

Interrupt-Anzeigeregister IFR. Die Interruptkontrolle findet in 3 Stufen statt. Im IFR können die möglichen Interrupt-Quellen (nämlich die 4 Steuerleitungen, die beiden Timer bei Nulldurchgang und das Schieberegister nach Abarbeitung) lediglich ein Flag setzen. Bit 7 ist logisch ODER auf die nachfolgenden Flags gesetzt und wird '1', sobald nur ein Flag '1' wird. Das ermöglicht eine schnelle Abfrage mit dem BIT-Befehl.

Interrupt Enable Register IER. Für jedes der vorgenannten Flags (Interruptquellen) kann durch Setzen seines Bits in diesem Register bestimmt werden, ob es einen Interrupt auslösen darf. Wenn ja, dann bringt das auslösende Ereignis das Ausgangssignal IRQ des Chips auf 'O' (dritte Stufe).

3. Die Ports als E / A-Leitungen

Hier sind die gleichen Grundfunktionen verwirklicht, die wir vom 6520 PIA und vom 6530 RRIOT kennen. Wegen seiner Verbreitung mit dem VIA stellen wir unsere Beispiele auf den Adressenbereich des AIM 65 ab.



Die Pins des Ports werden bitweise, also individuell, als Eingänge/Ausgänge (E/A oder I/0) als Verbindung zur Außenwelt geschaltet, indem man *zunächst* ein Bitmuster in das zugehörige Datenrichtungsregister DDRA (Adresse A003) oder DDRB (AOO2) einschreibt.

Für vorstehende Schaltung der Pins 7-5 als Eingänge und 4-0 als Ausgange sowie für die erwünschten Ausgangssignale programmiert man

```
LDA #$ 1F      FESTLEGUNG DER PINS FÜR E/A
STA DDRB       IM DATENRICHTUNGSREGISTER
LDA #$1A       ERWÜNSCHTE AUSGANGSSIGNALE 'HIGH' UND 'LOW'
STA ORB        IN DEN PORT
```

Entsprechend den Empfehlungen in unserem „Leitfaden für die Programmierung" haben wir die Eingänge für den BIT-Befehl auf die hohen Pins gelegt. Über die Prüfung und Umschaltung von Signalen ebendort.

Über die TTL-Kompatibilität und Belastbarkeit der Pins orientiere man sich in den Datenblättern. Daß PB7 als Output für Timer-kontrollierte Impulse Sonderfunktionen übernehmen und daß PB6 für Ereigniszählung dienen kann, wurde bereits erwähnt und findet weitere eigene Darstellung.

Während die Dinge für das Schreiben in die Peripherie übersichtlich und einfach liegen, sind für das Lesen einige Besonderheiten zu beachten, mit deutlichen Unterschieden für die A- und B-Seite des VIA. Zunächst ist zu erwähnen, daß jede Seite neben dem Output-Register (ORA oder ORB) zwei weitere Schaltungen aufweist, das Leseregister (Input Register IRA bzw. IRB) und das Lese-Latch. Was in diesem Leseregister steht und was damit auf den Datenbus gelangt, hängt nicht nur vom anstehenden Signal, sondern auch von der Funktion ab, die den Steuerleitungen zugewiesen wurde. Vorgreifend ist darauf hinzuweisen, daß die Steuerleitungen CA1 und CB1 durch Einschreiben in das Steuerleitungsregister PCR auf das Erkennen aktiver Übergänge geschaltet werden können (aufsteigende oder abfallende Impulsflanke) und daß in Abhängigkeit davon verschiedene Vorgänge im Chip auslösbar sind, unter anderem auch ein Handshake an dem korrespondierenden Pin CA2 bzw. CB2 als Ausgang (Handshake Control).

Und dann gibt es da noch die Latching-Kontrolle, von den beiden niederwertigsten Bit des Hilfsregisters ACR kontrolliert und unabhängig für die A- und für die B-Seite ausübbar. Das bedeutet: Man liest den Port so, wie er sich im Moment des Lesens befindet (das ACR-Bit ist in Normalstellung auf '0') oder so, wie er sich *vor* dem aktiven Übergang am CA1- bzw. CB1-Pin befunden *hatte* (ACR-Bit '1').

Damit nicht genug der Besonderheiten. Es sind ja Fälle denkbar, in denen ein Ausgangspin unter der zu treibenden Last unter die Spannungsschwelle absinkt, die mit 'high' erkannt wird, obwohl er mit 'high' beschrieben wurde. Im Leseregister B lesen wir in diesen Fällen einen Mix aus Pin-Ausgangs-Sollsignalen und tatsächlichen Eingangspegeln, solange das von CB1 gesetzte Interrupt-Flag 'high' ist. Sobald dieses Flag gelöscht ist, liest man die tatsächlichen Pegel an den Pins, 'low' und 'high'.

Nachzutragen ist: Für den Prozessor haben die Leseregister IRA, bzw. IRB dieselben Adressen wie die Ausgangsports. Sofern man nicht im Handshake-Betrieb arbeitet (s.u.), kann man die A-Seite gleichmäßig in den Adressen A001 und A00F auslesen. Das Latching eröffnet die Möglichkeit, die Eingangskombination nach dem aktiven Übergang mit derjenigen zuvor z.B. mit logisch EOR zu vergleichen. Dabei werden die Veränderungen erkannt.

Die vielen Kombinationen werden durch folgende Tabelle übersichtlicher:

SEITE	ACR-LATCH ENABLE BIT 1 FÜR CB1	BIT Ø FÜR CA1	IFR BIT 4 FÜR CB1	IM LESEREGISTER WIRD GELESEN:
A	X	0	X	AKTUELLER SIGNALPEGEL
A	X	1	X	SIGNALPEGEL VOR AKTIVEM ÜBERGANG AN CA1
B	0	X	0	TATSÄCHLICHER AKTUELLER PEGEL AN DEN PINS
B	0	X	1	OUTPUT AN ORB UND AKTUELLER PEGEL AN DEN INPUT-PINS
B	1	X	0	SIGNALPEGEL AN DEN PINS VOR AKTIVEM ÜBERGANG AN CB1
B	1	X	1	MIX AUS SOLL-OUTPUT AN ORB UND INPUT-SIGNALPEGEL VOR DEM AKTIVEN ÜBERGANG AN CB1

Das nachfolgende Beispiel initialisiert wie folgt: CB1 reagiert auf aufsteigende Impulsflanke, Port B ist als Eingang geschaltet mit Latching der Signale.

```
    LDA #$02        ERMÖGLICHE LATCHING AN DER B-SEITE
    STA ACR         HILFSREGISTER
    LDA #$10        CB1 AUF AUFSTEIGENDE FLANKE TRIGGERN
    STA PCR         STEUERREGISTER
    LDA #$00        PORT B ALS INPUT
    STA DDRB        DATENRICHTUNGSREGISTER
    ...
W   LDA #$10        MASKE FÜR CB1 INTERRUPT-FLAG
    BIT IFR         INTERRUPT FLAG REGISTER
    BPL W           WARTESCHLEIFE
    BNE XX          INTERRUPTBEARBEITUNG FÜR PORT B
```

In der vorstehenden Schleife W wird Bit 7 des IFR darauf geprüft, ob am Interfacebaustein irgendeine Interruptbedingung eingetreten ist, erst dann wird gezielt für CB1-Interrupt verzweigt. Wenn nur CB1 an diesem Baustein ein Flag setzen kann, programmiert man kürzer

```
    LDA #$10        MASKE
W   BIT IFR         DER BIT-BEFEHL VOLLZIEHT LOGISCH AND MIT DER MASKE
    BEQ W           WARTESCHLEIFE
```

Die häufig gebrauchten Techniken der Portbedienung dürften weitgehend bekannt sein. Z.B. Abfrage, ob sich irgendein Eingangssignal geändert hat:

```
    LDA PORT        ALTE SIGNALKOMBINATION
W   CMP PORT        WARTE, BIS SICH ETWAS ÄNDERT
    BEW W           WARTESCHLEIFE
```

An Ausgangsports lassen sich — unabhängig von den besonderen Fähigkeiten des VIA — Rechteckimpulse durch Flippen des niedrigsten Bits wie folgt erzeugen:

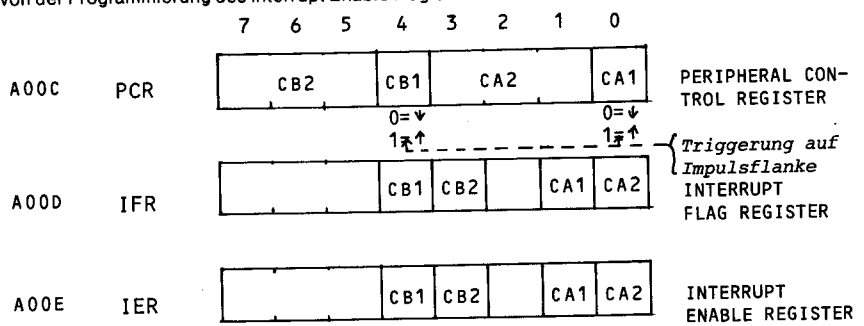
```
INC PORT        BIT0 AUF '1' SETZEN, WENN ES ZUVOR '0' WAR
DEC PORT        UND WIEDER AUF '0'
```

Für Tastaturabfragen u.ä. ist es geläufig, eine Rasterung von Gitterpunkten in der Weise vorzunehmen, daß man eine logische 'O' durch einen Sendeport shiftet (ASL oder LSR) und an einem Empfangsport abfragt, in welcher Bitposition die '0' empfangen wurde. Man nimmt die '0', weil unbeschaltete offene Leiterbahnen wie '1' senden.

Durch Beschaltung eines Ports mit z.B. 2 Stück 4 zu 16 Dekodern/Multiplexern kann die Zahl der möglichen Verbindungen zur Außenwelt ggfs. vervielfacht werden. Es sind natürlich auch andere Beschaltungen für das Multiplexing möglich, wie z.B. mit dem 4016 oder 74157. Mit einem Steuerbit könnte man 2x7 Datenbit umschalten.

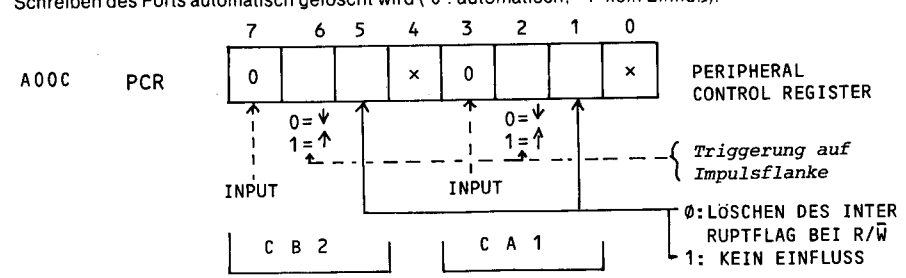
4. Impulsflanken an den Eingangs-Steuerleitungen

Die vier Pins CA1, CA2, CB1 und CB2 können im Steuerleitungsregister einzeln so programmiert werden, daß sie als Eingangsleitungen und in Reaktion auf steigende oder fallende Impulsflanken ihr korrespondierendes Interrupt-Flag im Interrupt-Anzeigeregister auf '1' setzen. Ob daraus nur eine Abfragemöglichkeit oder ein tatsächlicher Interrupt wird, hängt von der Programmierung des Interrupt Enable Registers IER ab.



CA1 und CB1 können bei aktivem Übergang am Eingang nur ihr Interrupt-Flag setzen. Für die B-Seite gilt zusätzlich, daß ein Lesen oder Schreiben des Ports das Interrupt-Flag automatisch löscht.

CA2 und CB2 werden durch Bit 3 = '0' im PCR bzw. Bit7 = '0' als Eingänge geschaltet. Bitposition 2 bzw. 6 im PCR legt die Art der Impulsflanke fest ('0' entspr. abfallend, '1' entspr. aufsteigend). Und die Bitposition 1 bzw. 5 bestimmt, ob das Interrupt-Flag durch Lesen oder Schreiben des Ports automatisch gelöscht wird ('0': automatisch, '1' kein Einfluß).



Beispiel: CB2 soll als Input bei aufsteigender Impulsflanke Interrupt auslosen. Ein Lesen/Schreiben des Port B soll in der ab 0F80 beginnenden Interruptroutine das Interrupt-Flag automatisch löschen:

```
LDA #$80      INITIALISIEREN DES IRQ-VEKTORS (AIM)
STA IRQV2     DIE ADRESSE IST A404
LDA #$0F
STA IRQV2+1
LDA #$40      CB2-KONTROLLE
STA PCR       STEUERLEITUNGSREGISTER
LDA #$88      INTERRUPTFREIGABE FÜR CB2. WEGEN BIT7='1' S.U.
STA IER
```

5. Die Steuerleitungen als Ausgang und Handshaking

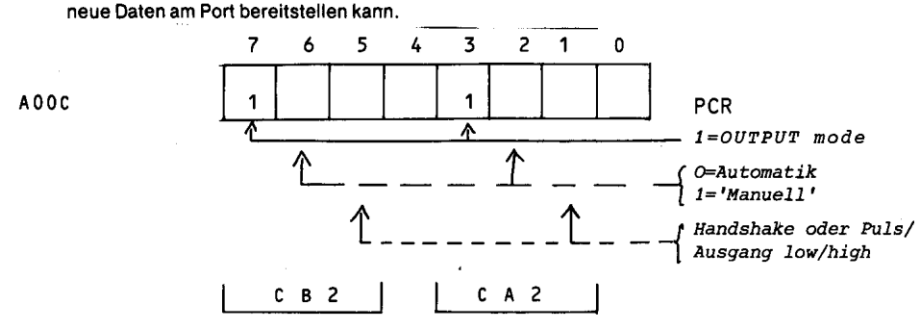
Handshaking ist beim asynchronen Verkehr zwischen Prozessor und Peripherieeinheit der Signalaustausch zum Zwecke der zuverlässigen Synchronisation von Eingabe oder Ausgabe über die Ports. Englisch spricht man von 'talker' und 'listener' für die betroffenen Einheiten. Vom Talker wird ein Signal 'data ready' und vom Listener eines 'data taken' erwartet. Beim 6522 VIA stehen für diesen Signalaustausch die bereits genannten 4 Steuerleitungen zur Verfügung.

CA1 und CB1 dienen als Input für die Erkennung aktiver Übergänge. Sie setzen Interrupt-Flags. Ist eine sendende Peripherieeinheit angeschlossen, so bedeutet ihr Signal an den Eingangspins 'ready to transmit to processor'. Handelt es sich dagegen um eine Ausgabeeinheit, so signalisiert sie dem Prozessor auf diesen Leitungen 'ready to receive' oder 'data taken'.

Im Handshake-Verkehr dienen dann CA2 oder CB2 als Ausgabeleitungen, um einer sendenden Einheit mitzuteilen 'data taken' oder um einer Ausgabeeinheit anzukündigen 'data ready for transmission'.

CA2 und CB2 werden als Ausgänge geschaltet, indem man in Bitposition 3 bzw. 7 des PCR eine '1' einschreibt. Für jede Seite des VIA gibt es 4 Möglichkeiten zur Erzeugung des Ausgangssignales. Die eleganteste finden wir nur auf der A-Seite, das automatische Handshaking: Sobald man (bei der Eingabe in den Prozessor) in der Prüf- oder Interruptroutine den Port A liest, geht CA2 in automatischer Beantwortung auf 'low'. Das bedeutet 'data taken'. Im handshake mode geht CA2 erst dann wieder auf 'high', wenn die Peripherie durch aktiven Übergang an CA1 meldet 'new data ready'. Daneben gibt es auch eine Betriebsart, in der CA2 oder CB2 mit einem kurzen Impuls quittieren.

Ähnlich sinnvoll liegen die Verhältnisse bei der Ausgabe an die Peripherie: Ein Beschreiben des Ports A oder B setzt CA2 bzw. CB2 auf 'low'. Damit wird 'data ready for transmission' signalisiert. Das Ausgangssignal verharrt auf diesem Pegel, bis von draußen über CA1 (CB1) 'data taken' gemeldet wird. Erst jetzt wird das Interrupt-Flag gesetzt, damit der Prozessor neue Daten am Port bereitstellen kann.



Beispiel: Am Port A sollen Daten im Interrupt-Betrieb empfangen werden (Routine in 0F00 beginnend). Die sendende Einheit meldet mit aufsteigender Flanke an CA1 'data ready' und will einen kurzen Impuls über CA2 für 'data taken' erhalten:

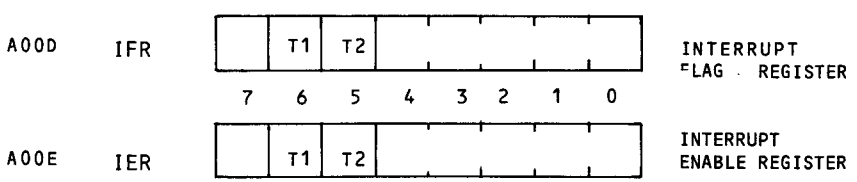
```
LDA #$00        INITIALISIERE INTERRUPT-VEKTOR
STA IRQV2
LDA #$0F
STA IRQV2+1
LDA #$0B        CA1 AUF AUFSTEIGENDE FLANKE, CA2 AUF PULSAUSGABE
STA PCR         STEUERLEITUNGSREGISTER
LDA #$82        ERMÖGLICHE INTERRUPT FÜR CA1
STA IER
```

Unabhängig von solchem Handshake-Betrieb können CA2 und CB2 unter Kontrolle des Programmes auch auf definierte Pegel 'high' oder 'low' gesetzt werden (manual output mode) um z.B. äußere Multiplexer umzusteuern. Bei AIM 65 wird z.B. über CA2 das Casetten-Interface von Eingabe auf Ausgabe umgeschaltet.

Die Vielseitigkeit der Ausgangssteuerung läßt für die Bedienung wohl kaum Wünsche offen, zumal noch eine Ergänzung hinzukommt: Im handshake mode an Port A geht CA2 auf 'low', wenn man den Port in Adresse A001 liest. Wenn neue Daten noch nicht bereitstehen, könnte das in zeitlicher Überschneidung zu Mißverständnissen führen. Es wurde daher für Zwischenprüfungen die Adresse A00F geschaffen, deren Einlesung oder Beschreibung die Handshake-Steuerung nicht verändert. Damit ist auch die Option eröffnet, den Port gemischt als Ein- und Ausgang zu fahren. Die Steuerung entsprechender Pins eines Ausganges oder die Prüfung eines zweiten Einganges beeinflußt dann nicht die ebenfalls angeschlossene Handshake-Einheit.

6. Die Timer / Zähler

Das VIA trägt zwei Timer, T1 und T2, von je 16 Bit Breite. Anders als bei den Timern z.B. des 6532 RIOT und des 6530 RRIOT (im KIM) arbeiten sie nicht mit einstellbaren Teil-oder Zählgeschwindigkeiten (z.B. :8,:1024), sondern zählen die 2 Byte breite Vorgabe mit jedem Maschinentakt um 1 herunter. Beide Timer setzen bei ihrem Nulldurchgang ein eigenes Interrupt-Flag im Interrupt-Anzeigerregister (IFR) und können auch wahlweise einen Interrupt auslösen, und zwar unter Kontrolle des im Interrupt Enable Register (IER) hinterlegten Bitmusters.



Den Timern ist gemeinsam, daß sie für die CPU definierte Zeit- und Verzögerungsintervalle erzeugen können. Ihre eigentliche Leistungsfähigkeit geht weit darüber hinaus. Timer T1 kann man etwa als Erzeuger von vielfältigen Rechteckimpulsen an Pin PB7 charakterisieren, T2 als Intervalltimer, Zähler und Kontrolleur für das serielle Schieberegister. Die Sonderfunktionen der Timer und der Pins PB7 und PB6 werden dabei vom Hilfsregister ACR kontrolliert.

Den Timern ist weiterhin die Zuordnung von Vorspeicher-Latches gemeinsam. T1 hat Latches für beide Bytes, T2 nur ein Latch für das niederwertige Byte.

6.1 Der Timer T1

Diesem Timer sind 4 Registeradressen zugeordnet, bei dem Schreiben und Lesen teilweise jedoch eine unterschiedliche Bedeutung haben:

A004 T1L-L LOW ORDER LATCH	SCHREIBEN	A005 T1L-H HIGH ORDER LATCH T1C-H HIGH ORDER COUNTER TRANSFER T1L-L TO T1C-L
T1C-L LOW ORDER COUNTER	LESEN	T1C-H HIGH ORDER COUNTER

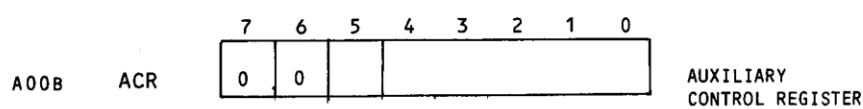
A006 T1L-L LOW ORDER LATCH	SCHREIBEN UND LESEN	A007 T1L-H HIGH ORDER LATCH

Zunächst ist festzuhalten, daß man in den niedrigen Zählerteil nicht direkt hineinschreiben kann, sondern nur in seinem Vorspeicher T1L-L. Das Laden dieses Zählerteils aus dem Vorspeicher erfolgt erst in dem Augenblick, da man in das hohe Zählerbyte, Adresse A005, hineinschreibt. Damit beginnt der 'count down' gleichzeitig an der gesamten 16-Bit-Vorgabe. Nach Nulldurchgang des Timers wird sein Interrupt-Flag gesetzt.

Ein Beispiel für einfache Zeitverzögerung von 160 usec mit Interrupt-Enable, Interruptroutine in 0F00:

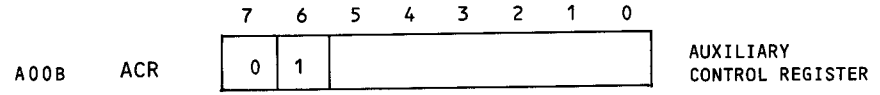
```
LDA #$00      INITIALISIERE INTERRUPT-VEKTOR
STA IRQV2
LDA #$0F
STA IRQV2+1
LDA #$80      EINFACHES TIME OUT
STA ACR       HILFSREGISTER
LDA #$C0      INTERRUPTM=GLICHKEIT FÜR T1
STA IER       INTERRUPT ENABLE REGISTER
LDA #$A0      TIMERVORGABE LOW, 160 µSEC
STA T1L-L     INS LATCH
LDA #$00      VORGABE HIGH
STA T1C-H     STARTE TIMER DURCH SCHREIBEN IN DEN ZÄHLER (A005)
```

Zu diesem einfachen time-out oder 'one shot', bei der der Timer nur einmal die Zählervorgabe durchläuft, gehört folgendes Bitmuster im ACR:



Nach seinem Nulldurchgang wird der Timer weiter mit dem Systemtakt 02 heruntergezählt, so daß man ggfs. die Zeit seit dem Nulldurchgang erfassen kann. Das Interrupt-Flag wird dadurch gelöscht, daß man den Timer in T1C-L (AOO4) liest oder in T1C-H (AOO5) neu beschreibt.

Neben diesen einfachen time-out gibt es weitere drei Betriebsweisen. Da ist zunächst 'free running' zu erwähnen, Bit 6 im ACR ist gesetzt:



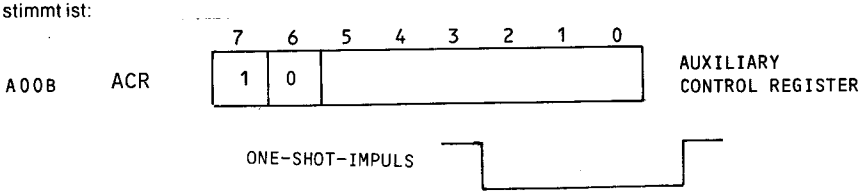
Nach jedem time-out wird der Zähler automatisch mit dem Inhalt der Vorspeicher T1L-L und T1L-H (A006, A007) nachgeladen. Das Interrupt-Flag muß man dann aber per Programm zurücksetzen, indem man z.B. T1C-L liest. Damit sind wir auf eine weitere Besonderheit gestoßen: Der niedrige Zählerteil kann jederzeit gelesen werden, geladen wird er aber immer aus dem Vorspeicher-Latch. Der hohe Zähleranteil kann direkt gelesen und beschrieben werden, er wird automatisch aus dem Latch T1L-H nachgeladen, wenn mit ACR6='1' 'free running mode' gesetzt ist. Und damit wird auch gleich der Nutzen der Vorspeicher ersichtlich:

Bei 'free running' kann man den Timer irgendwann zwischenzeitlich mit der Zeitkonstanten für die nächste Phase vorladen, während er noch an der alten Phase herunterzählt. Nimmt man gar keine Veränderung der Werte in den Vorspeichern vor, so werden die Zeitabschnitte gleich lang, und es liegt eine Form der Selbstverwaltung im VIA vor. In nachfolgendem Beispiel nehmen wir einmal an, der Interruptvektor sei bereits geladen und die Zeitabschnitte sollten alle gleich sein. Wir programmieren dann:

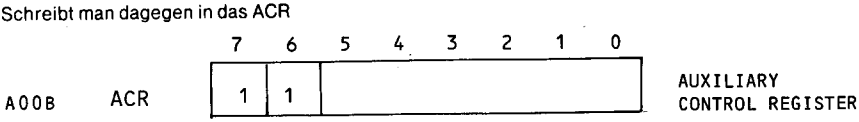
```
LDA #$40      FREE RUNNING
STA ACR       HILFSREGISTER
LDA #$XX      LADE NIEDRIGEN VORSPEICHER
STA T1L-L     INS LATCH
LDA #$YY      VORGABE FÜR DAS HÖHERE BYTE
STA T1C-H     STARTE TIMER UND LADE ZUGLEICH LATCH T1L-H
```

In der Interrupt-Routine setzen wir das Interupt-Flag durch LDA T1C-L zurück.

Die beiden vorgenannten Betriebsweisen, die nur das Interrupt-Flag beeinflussen, werden ergänzt durch das 'PB7-Enable' (Bit 7 im ACR='1'). PB7 dient jetzt als Ausgang für Rechteckimpulse, die vom Timer T1 regiert werden. Die nachfolgende Bitkombination im ACR erzeugt einen einmaligen Rechteckimpuls an PB7, dessen Impulsweise durch die Timervorgabe bestimmt ist:



Schreibt man dagegen in das ACR

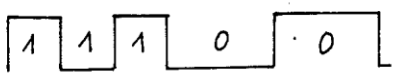


so haben wir wieder „free running" mit der Besonderheit, daß nach jedem time-out eine Phasenumkehr an PB7 eintritt, wodurch eine Rechteckschwingung mit beeinflußbarer Phasendauer entsteht. Die Magnetbandformate des AIM 65 werden auf diesem Wege erzeugt. Nicht ohne besonderen Grund ist PB7 bei diesem Gerät mit dem Audio-Interface ver-

bunden. Aber auch unser Programm HAMMING WAY in Heft 6 macht von den Möglichkeiten Gebrauch. Man beachte dort das Beispiel: Erzeugen einer Rechteckschwingung mit gleichmäßigem Burst und nachfolgender veränderlicher Halbschwingung für "0" oder "1".

Das nachfolgende kleine Unterprogramm dient der Wechselphasenkodierung. Nicht mehr der Signalpegel und auch nicht die Abfolge von Impulsen ist für die Kodierung einer "1" oder "0" entscheidend, sondern allein die Zeit zwischen den Phasenwechseln.

Das Unterprogramm sendet ein im Akku übergebenes Byte an PB7 als Ausgang. Es setzt voraus, daß schon initialisiert und daß der Timer schon bestartet wurde:




```
SUB   LDY #$08     FÜR 8 BIT
SU1   ASL A        GEWINNE EIN CARRY, WENN '1'
      PHA          RETTE DEN REST DES BYTE
      LDA #$50     PHASENLÄNGE 80 µSEC FÜR '1'
      BCS WEI      ÜBERSPRINGE FÜR BIT='1'
      LDA #$A0     PHASENLÄÄNGE FÜR '0' VON 160 µSEC
WEI   STA T1L-L    IN DEN VORSPEICHER DAS HÖHERE BYTE BLEIBT BEI '00'
PAT   BIT IFR      WARTE AUF TIME-OUT
      BVC PAT
      LDA T1L      LÖSCHE FLAG
      PLA          HOLE REST DES BYTE
      DEY
      BNE SU1      NÄCHSTES BIT
      RTS
```

6.2 Der Timer T2

Diesem Timer sind 2 Registeradressen zugeordnet, die wie beim T1 beim Lesen und Schreiben eine unterschiedliche Bedeutung für das niedrige Byte haben.

A008		A009
T2L-L LOW ORDER LATCH	SCHREIBEN	T2C-H HIGH ORDER COUNTER TRANSFER T2L-L TO T2C-L
T2C-L LOW ORDER COUNTER	LESEN	T2C-H HIGH ORDER COUNTER

Auch hier kann der untere Zählerteil nicht direkt beschrieben werden, er wird in dem Moment aus seinem Vorspeicher T2L-L geladen, wo man in das obere Zählerbyte T2C-H einschreibt. T2 arbeitet im 'one shot mode' als Intervalltimer, wenn Bit 5 im ACR = '0'. Bei seinem Nulldurchgang setzt er einmalig sein Interrupt-Flag und zählt dann weiter von 'FFFF' herunter. Das Interrupt-Flag wird durch Lesen von T2C-L oder neues Beschreiben von T2C-H gelöscht. Einen 'free running mode' gibt es für T2 nur in einer bestimmten Betriebsart in Verbindung mit dem Schieberegister.

Setzt man Bit 5 im ACR auf "1", so fungiert T2 als "Count-Down-Zähler" für an PB6 angelegte abfallende Impulsflanken. Der Zähler ist mit einer Vorgabe zu laden. Das Interrupt-Flag erscheint, wenn die Vorgabe mit Nulldurchgang des Zählers verbraucht ist.

T2 kann unter gewissen Bedingungen auch als Taktgenerator für das Schieberegister SR eingesetzt werden (s.u.), dann ist allerdings nur der untere Zählerteil in Aktion.

7. Das Schieberegister

Es dient dem seriellen Datenverkehr über Pin CB2 als Ein- oder Ausgabe. Intern besteht eine Verkoppelung mit einem Modulo-8-Zähler. Auf dem VIA sind für die Eingabe (ACR-Bit4*'0') drei und für die Ausgabe (ACR-Bit4='1') vier Betriebsarten vorgesehen. Bei der seriellen Datenübertragung kommt es wesentlich auf ein definiertes Timing zwischen Sender und Empfänger an. Pin CB1 übernimmt dabei neue Funktionen, die vom ACR gesteuert werden. Man kann dort den Takt einer externen Quelle anlegen (z.B. des Senders) oder einen intern erzeugten Takt für einen Empfänger über CB1 aussenden.

Als Takt an CB1 stehen zur Verfügung: a.) die Systemuhr 02, deren effektiver Takt mit 0,5 MHz arbeitet, b.) die Zeitvorgabe im niedrigen Teil des Timers 2 (T2L-L) mit 3900 Baud als niedrigster Geschwindigkeit, c) z.B. Clockimpulse, die man im Timer 1 erzeugt und aus PB7 auf CB1 führt. Im letzteren Fall sind langsame Übertragungsraten oberhalb 8 Baud erzielbar.

Die hohen Geschwindigkeiten unter a) und besonders b) sind in erster Linie für den Datenverkehr zwischen Prozessorsystemen gedacht. Beim 'distributed processing' ist es also innerhalb eines festgelegten Protokolls möglich, z.B. den PET mit dem AIM 65 via 6522 zu verbinden. Es bedarf dann nicht einer Stromschleife oder der Herstellung kompatibler Magnetbandaufzeichnungen.

Hinsichtlich des Taktes bei serieller Datenübertragung beachte man in den Diagrammen des 6522-Datenblattes sehr genau, wann mit aufsteigender und wann mit abfallender Impulsflanke Daten übernommen werden. Im übrigen kann CB2 als Ausgabe für den Ruhezustand im PCR sowohl auf 'high' wie auf 'low' programmiert werden.

8. Interruptkontrolle

In einigen Beispielen dieses Artikels wiesen wir schon auf die 3 Stufen des Interrupts hin. Den möglichen Interruptquellen des VIA (Steuerleitungen, Timer und Schieberregister) sind im Interrupt-Anzeigeregister IFR Flags zugeordnet. Sobald ein Flag auf '1' geht, wird dort auch Bit7 logisch ODER auf '1' gesetzt. Das ermöglicht dem Prozessor mit dem BIT-Befehl eine schnelle Abfrage, ob ein Interrupt vom betreffenden Chip ausging. Wenn ja, dann kann in diesem IFR durch logische oder Verschiebebefehle die Interruptquelle im einzelnen schnell ausgemacht werden.

Eine hier angezeigte Interruptbedingung muß nicht notwendig zum Verlassen des im Vordergrund laufenden Maschinenprogramms führen. Welche Bedingungen — oder ob gar keine — Interrupt auslösen dürfen, wird durch Einschreiben in das IER, das Interrupt Enable Register festgelegt. Wenn man den Hardware-Interrupt IRQ zuläßt, dann muß man rechtzeitig auch den Interruptvektor geladen haben.

		7	6	5	4	3	2	1	0	
A00D	IFR	IRQ	T1	T2	CB1	CB2	SR	CA1	CA2	INTERRUPT FLAG REGISTER
A00E	IER	SET /CLR	T1	T2	CB1	CB2	SR	CA1	CA2	INTERRUPT ENABLE REGISTER

Elegante Möglichkeiten sind vorgesehen, um im Interrupt Enable Register Interruptmöglichkeiten einzelbitweise zu- oder abzuschalten. Dabei hat Bit7 des einzuschreibenden Bytes eine Steuerfunktion. Ist es '0', so löscht jede nachfolgende '1' die Interruptmöglichkeit nur an entsprechender Stelle und umgekehrt wird mit Bit7*'1' einzelbitweise Interruptmöglichkeit hinzugeschaltet.

R.L.

Der AIM-ASSEMBLER

E: Rockwell offers an Assembler ROM as option for the AIM 65. In the following we cover the requirements for input to the assembler and its several forms of output, the symbol table, the assembler list and the generated object code.

1. Begriffliches

Ein Assembler besteht aus:

a) Einer maschienorientierten symbolischen Programmiersprache und
b) einem Programm zur Umwandlung (Übersetzung) dieser Assemblersprache in die Maschinensprache (object code).

Dieser Aufsatz behandelt die Assemblersprache. Das Übersetzungsprogramm wird für den AIM 65 in Form eines R2332-Festwertspeichers als Option angeboten. Für den PET wurde ein mögliches in BASIC formuliertes Assemblerprogramm in Heft 6 abgedruckt (PET 6502-Assembler).

Als digitales Gerät kann der Computer nur rein binäre Information in Form von Opcodes und Adressen verarbeiten. Die Verwendung von hexadezimalem Code in Programmierhandbüchern, Programm-Listings und an den Eingabetastaturen stellt bereits ein begriffliches Entgegenkommen bzw. eine Bequemlichkeit für den Bediener dar. Notwendig ist sie nicht. Wir sehen bei der Formulierung von Maschinen-Unterprogrammen in der Sprache BASIC, daß man dort dezimal kodierte Instruktionen verwendet. Der Gebrauch von Oktalziffern (Basis 8) ist dagegen seltener, aber ebenso möglich.

Wer von den kleineren Entwicklungsgeräten wie etwa KIM herkommt, fand dort häufig die Begriffsverwendung 'Programmieren in Assemblersprache'. Ganz richtig ist das nicht, denn ohne Hilfsprogramm muß der Bediener jeden der als Merkhilfe verwendeten mnemonischen Opcodes manuell in die Maschinensprache umsetzen und auch selber die Verwaltung der benutzten Adressen vornehmen. Der Bediener vollzieht dort also selber das Assemblieren.

Beim Programmieren unter I-Command des AIM-Monitors tritt bereits eine wesentliche Entlastung ein: Es dürfen mnemonische Opcodes eingetastet werden, die Adressierungsart des Operanden ist durch seine Schreibweise beeinflußbar und ein Zuordnungszähler rückt je nach Länge der Maschineninstruktion um 1-3 Byte weiter.

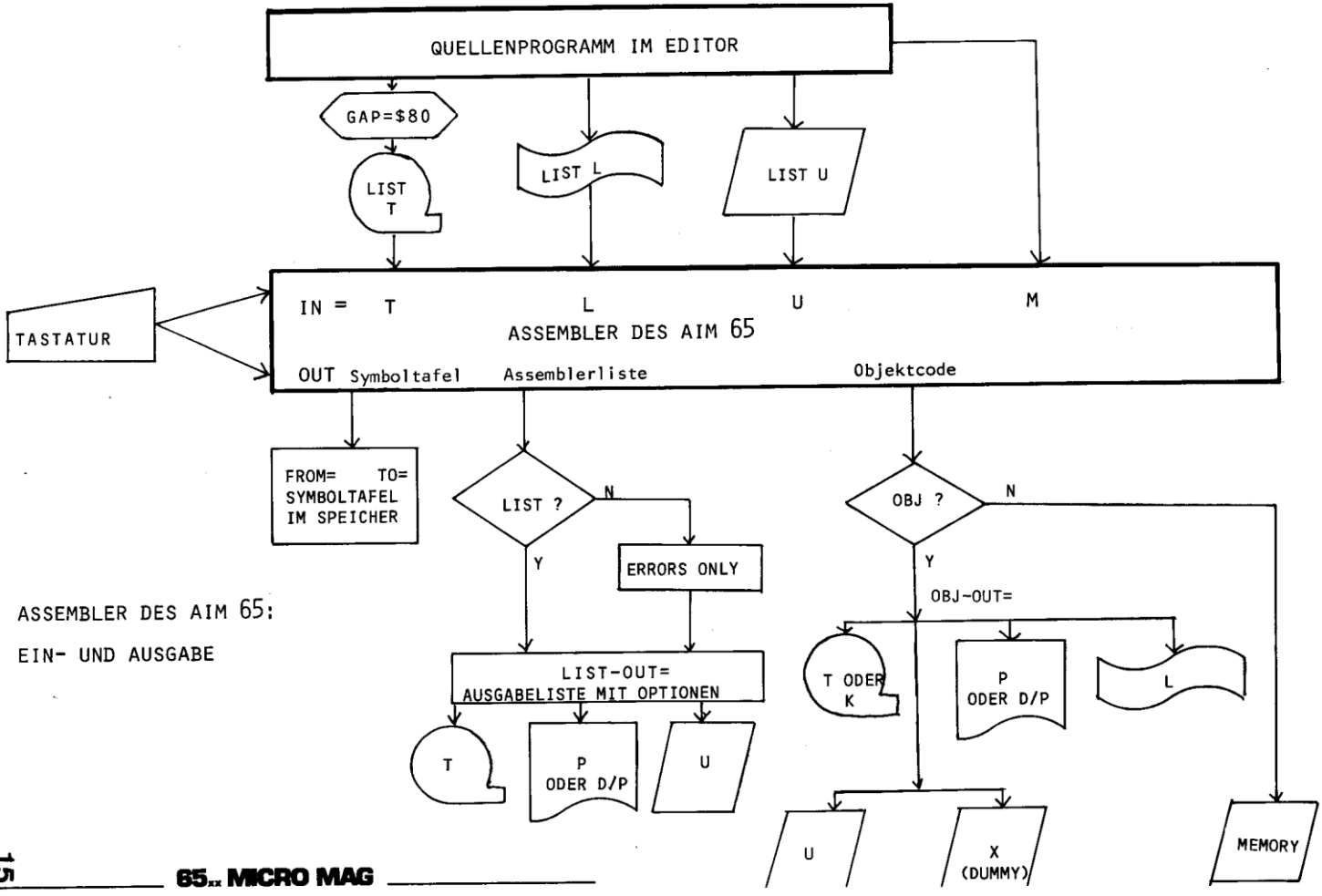
Der Assembler bietet weitere Verbesserungen: Mit einer einzigen Anweisung am Programmbeginn kann festgelegt werden, für welchen Adressenbereich das Quellenprogramm zu berechnen ist (Verschieblichkeit). Es dürfen weiterhin gut merkbare symbolische Namen für Konstante, Arbeitsbereiche und Einsprungspunkte benutzt werden. Das Assemblerprogramm hat ferner eine gewisse Rechenhaftigkeit zur Bewertung von Rechenausdrücken und es enthält zahlreiche Anweisungsmöglichkeiten für das Erzeugen einer Ausgabeliste als Dokumentation oder Kontrolle. — Anders als beim Assembler größerer Geräte erfolgt das Programmieren nach wie vor byteweise.

2. Elemente der Assemblersprache

a.) In der mnemonischen Assemblersprache kodierte Befehle des Quellenprogrammes. Der Befehlssatz (56) entspricht dem des Mikroprozessors, auch hinsichtlich der 13 Adressierungsarten für die Operanden. Jeder Befehl im Quellenprogramm führt 1:1 zu einem Maschinenbefehl (im Gegensatz zu problemorientierten Sprachen wie etwa BASIC oder FORTRAN).

b.) Bemerkungen sind Erläuterungen zum Programm, die im Umwandlungsprotokoll (Assemblerliste) erscheinen, aber nicht in Maschinensprache übersetzt werden.

c.) Assembleranweisungen, die in der Phase der Umwandlung Hilfsfunktionen steuern und ebenfalls nur auf der Liste protokolliert werden und nicht zu Maschinenbefehlen führen. Zu dieser Gruppe gehören:



ASSEMBLER DES AIM 65:

EIN- UND AUSGABE

* Deklaration von Konstanten mit Wert und symbolischen Namen,
* Deklaration von Speicheradressen als Arbeitsspeicher, Eingangs- oder Referenzpunkt unter einem symbolischen Namen (label),
* Operatoren für Berechnungen in der Umwandlungsphase (+,—, < , >),
* Setzen des Zuordnungszählers,
* Die Assemblerein- und -ausgabe beeinflussende Anweisungen.

3. Das Arbeiten mit dem Assembler

3.1. Die Eingabeseite

Der Assembler fordert, daß der Benutzer sein Programm mit Hilfe des Text-Editors bereits in der Assemblersprache neu formuliert hat. Das Programmieren beginnt also nicht, wenn man die Taste 'N' drückt, es muß vorher stattgefunden haben. Die beifolgende Graphik verdeutlicht den möglichen Input für den Assembler: Verständigt man ihn per Tastatur mit 'M', so arbeitet er am Inhalt des im Editor speicherresidenten Quellenprogrammes. Der Input kann aber auch von anderen Quellen herrühren, auf die das im Editor entwickelte Quellenprogramm im Vorwege gedumpt, abgesetzt worden ist, also Magnetband (T), Lochstreifen (L) oder USER (U). An den letztgenannten möglichen Programmquellen erkennt man die vielseitige Anlage des Assemblers:

Eine Programmzeile im Editor-Quellenprogramm belegt viele Bytes. Maschinenprogramme benötigen je Befehl jedoch nur 1-3 Bytes. Wenn man also von außen über Magnetband o.ä. zugeführte Quellenprogramme verwertbar macht, so benötigt man für diese nur den Eingabepuffer im Speicher. Das zu generierende Maschinenprogramm kann also umso umfangreicher werden.

Besonderheit: Eingabe vom Magnetband erfordert Fernsteuerung des Laufvorganges (remote controlled) und ausreichende Bandlücken (GAP in $A409=$80) bei der davorliegenden Ausgabe auf das Magnetband.

3.2. Die Ausgabeseite

Die Vielseitigkeit des Assemblers betrifft auch die Ausgabeseite (siehe Graphik). Der noch wenig geübte Leser sei darauf hingewiesen, daß die Verwendung von symbolischen Adressen im Quellenprogramm zur Zeit der Assemblierung zur Anlage einer SYMBOL TABLE führt, in der jeder Name 8 Bytes beansprucht. Für diese Tabelle ist von der Tastatur her eine geeignete Speicherzuweisung vorzunehmen (mit FROM= .... TO= ....), die eine Überschneidung mit dem etwa im Editor noch speicherresidenten Quellenprogramm oder auch mit dem im Speicher abzulegenden Objectcode vermeidet.

Als zeitweiliger Katalog der im Quellenprogramm verwendeten Symbole belastet die SYMBOL TABLE den Speicher also während der Generierungsphase. Je mehr Symbole, desto mehr Speicher. Ggfs. tritt ein Überlauf ein (SYM TBL OVERFLOW). Wegen der festen Belegung der Speicher-Pages 0 und 1 muß die SYMBOL TABLE oberhalb $0200 angelegt werden. Man identifiziere das FROM= und TO= also keinesfalls mit dem Platz des zu erzeugenden MASCHINENPROGRAMMES.

Die Graphik zeigt weiter, daß von der Tastatur her sowohl die Assemblerausgabeliste wie auch der erzeugte Objectcode auf verschiedene Ausgabekanäle gerichtet werden können. Zur Aufnahme der Ausgabeliste, auf deren Gestaltung wir noch zurückkommen, sind vorgesehen Display/Printer, Printer, Magnetband oder Peripherie des USER.

Das wichtigste Produkt des Assemblers ist schließlich das Maschinenprogramm. Ohne Speicherbeaufschlagung kann es auf Display/Printer, auf den Drucker allein, auf Magnetband, Lochstreifen, auf USER oder auf eine Dummy Device (keinerlei Ausgabe) gerichtet werden. Nur bei OBJ?=N gelangt es in den Speicher.

Die vielen Ausgabemöglichkeiten verdeutlichen: Wenn man sehr lange Quellen- und Maschinenprogramme hat, dann sollte man das Quellenprogramm vom Tonband, Lochstreifen o.ä. einlesen und auf ähnliche Medien ausgeben. In diesem Falle kann fast der gesamte Speicher des AIM 65 für die SYMBOL TABLE verwendet werden. M. a. W.: Der Text-Editor und die SYMBOL TABLE können speichermäßige Hindernisse darstellen, wenn man nicht ausreichend geplant hat.

3.3 Struktur des Quellenprogrammes

Wesentliche Bestandteile des Quellenprogrammes sind:

a) Setzen das Zuordnungszählers auf den gewünschten Anfangswert,
b) Das eigentliche Programm und
c) Die .END-Anweisung.

Der Zuordnungszähler kann dabei wahlweise mit einer hexadezimalen, binären dezimalen oder oktalen Zahl geladen werden. Der Assembler nimmt die benötigte Umrechnung vor. Die Anweisung .END ist obligatorisch.

Für das Setzen des Zuordnungszählers einige Beispiele:

```
EDITOR              ASSEMBLERLISTE          KOMMENTAR
*=33                ==0021 *=33             DEZIMAL
*=$33               ==0033 *=$33            HEXADEZIMAL
*=@11               ==0009 *=@11            OKTAL
*=%11               ==0003 *=%11            BINÄR
*=$CCCC+1           ==CCCD *=$CCCC+1        AUSDRUCK:HEXA UND DEZIMAL
```

3.4. Befehlssatz und Adressierungsarten

Für die Formulierung des Quellenprogramms kann auf die klaren Ausführungen im Abschnitt 5.7. des AIM-Anwenderhandbuches verwiesen werden. Wir geben anbei ein Beispiel mit allen 13 Adressierungsarten und verwenden dabei auch schon Symbole und Ausdrücke (wie +1, +2).

```
QUELLENPROGRAMM     ASSEMBLERLISTE          KOMMENTAR
LAB1=$4567          ==0000 LAB1=$4567       DEKLARATION
LAB2=$03            ==0000 LAB2=$03         DITO
SYMB=$AB            ==0000 SYMB=$AB         DITO
*=$400              ==0000 *=$400               ZUORDNUNGSZÄHLER
START LDX #SYMB     ==0400 START                LABEL
                    A2AB   LDX #SYMB            DIREKTOPERAND

LDA LAB2            A503   LDA LAB2          ZEROPAGE
LDA LAB2,X          B503   LDA LAB2,X        ZEROPAGE,X
LDX LAB2,Y          B603   LDX LAB2,Y        ZEROPAGE,Y
LDA (LAB2,X)        A103   LDA (LAB2,X)      (INDIRECT,X)
LDA (LAB2),Y        B103   LDA (LAB2),Y      (INDIRECT),Y
LDA LAB1            AD6745 LDA LAB1          ABSOLUTE
LDA LAB1,X          BD6745 LDA LAB1,X        ABSOLUTE,X
LDA LAB1+1,Y        B96845 LDA LAB1+1,Y      ABSOLUTE,Y
```

```
DEX                     CA      DEX             IMPLIED
BPL START               10E8    BPL START       RELATIVE
ZIEL JMP START+2        ==0418  ZIEL
                        4C0204  JMP START+2      ABSOLUTE
JMP (ZIEL)              6C1804  JMP (ZIEL)      ABS. INDIRECT
ASL A                   0A      ASL A           AKKU
```

3.5 Deklaration von Symbolen

Im vorstehenden Beispiel wurden bereits Symbole verwendet: LAB1, LAB2, SYMB, START und ZIEL. Die ersten drei erhielten ihre Wertzuweisung (in 2 Bytes) mit '=' als Deklaration während der Initialisierung. START und ZIEL sind Labels, die ihren Wert durch den jeweiligen Stand des Zuordnungszählers während PASS1 erhalten.

Bei der Wertzuweisung mit '=' darf auf das Gleichheitszeichen ein Rechenausdruck folgen, der Bestandteile auch aus verschiedenen Zahlensystemen enthält.

Wird ein Symbol nicht als Adresse, sondern als Rechenkonstante von einem Byte Länge deklariert, so wird ihr Wert gleichwohl in 2 Byte eingetragen, wobei MSB=00. Wir sehen das bei der Überprüfung der SYMBOL TABLE und auch an der Wirkung der truncate operator < und > , die zu einer Zeropage-Adressierung führen.

Zur Syntax der Deklarationen: Ein Symbol oder Label muß in den Programmzeilen immer als erstes auftauchen. Die 56 mnemonischen Opcodes sowie die 'reservierten Symbole' A, X, Y, S und P dürfen nicht benutzt werden. Ein Symbol darf bis zu 6 Zeichen haben, das erste muß alphabetisch sein.

```
QUELLENPROGRAMM         ASSEMBLERLISTE              KOMMENTAR
CONST=$56               ==0000 CONST=$56            DEKLARATION
*=$700                  ==0000 *=$700               ZUORDNUNGSZÄHLER
LDA #CONST              ==0700 A956   LDA #CONST
LDA CONST                      A556   LDA CONST    ZEROPAGE-ADDR.
LDA <CONST                     A556   LDA <CONST   LSB DES SYMBOLS
LDA >CONST                     A500   LDA >CONST   MSB DES SYMBOLS
LDA <CONST+1                   A557   LDA <CONST+1  ADDITION VON 1
```

3.6 Speicherplatzreservierung

Ohne Wertzuweisung erfolgt sie durch einfaches Setzen eines Labels unter der Adresse des Zuordnungszählerstandes und ggfs. nachfolgendes Hochzählen oder Verändern der Sternadresse (Z. B. *=*+2). Die Monitor-Listings enthalten dafür viele Beispiele.

Eine Zeile, die nur einen Label enthält, ist gültig. Der Zuordnungszähler rückt nicht vor, so daß unter der erreichten Adresse auch mehrere Namen eingetragen werden können:

```
QUELLENPROGRAMM         ASSEMBLERLISTE      KOMMENTAR
*=$700                  ==0000 *=$700       ZUORDNUNGSZÄHLER SETZEN
M1                      ==0700 M1           LABEL
M2 *=*+2                ==0700 M2           2. LABEL FÜR DIESE ADR.
                                            UND HOCHZÄHLUNG UM 2
M3                      ==0702 M3           LABEL
```

Speicherplätze werden — wahlweise auch mit Namen — durch die Anweisungen .BYTE, .WORD und .DBYTE reserviert und erhalten die nachfolgenden Parameter als Wert zugewiesen. .WORD und .DBYTE reservieren 2 Bytes, die Anordnung der eingegebenen Parameter ist zwischen diesen beiden entgegengesetzt. Die einzutragenden Parameter bzw. Parameterpaare können innerhalb einer Anweisung durch Komma verkettet sein, so daß man ganze Tabellen eingeben kann. Bei der Assemblerliste ist zu beachten, daß nach den beiden ersten übersetzten Zeichen zunächst der volle Quellentext abgedruckt wird. erst danach folgen die weiteren generierten Zeichen. Für Zeichenketten in ASCII sind die Assembleranweisungen .OPT GEN und .OPT NOG zu beachten.

Der Wert von Parametern darf sich auch aus Ausdrücken ergeben.

```
QUELLENPROGRAMM          ASSEMBLERLISTE          KOMMENTAR
.OPT GEN                 ==0000 .OPT GEN         ALLE ASCII ZCH. DRUCKEN
*=$700                   ==0000 *=$700           ZUORDNUNGSZÄHLER SETZEN
RES1 .BYT $1             ==0700 RES1             LABEL
                         01     .BYT $1          WERT UND TEXT
RES2 .BYT %11            ==0701 RES2             LABEL
                         03     .BYT %11         BINÄR=3
RES3 .BYT 1,2,3,4        ==0702 RES3             LABEL
                         01     .BYT 1,2,3,4     DEZIMALE WERTE
                         02
                         03
                         04
RES4 .BYT $A+%11         ==0706 RES4             LABEL
                         0D     .BYT $A+%11      10+3=13, ENTSPR. 0D IN HEX
RES5 .BYT 'AB','CD'      ==0707 RES5             LABEL
                         4142   .BYT 'AB','CD'   4 ASCII-WERTE
                         4344                   ALLE GENERIERT D. .OPT GEN
RES6 .WOR $ABCD          ==070B RES6             LABEL
                         CDAB   .WOR $ABCD       REIHENFOLGE LOW-HIGH ORDER
RES7 .DBY $ABCD          ==070D RES7             LABEL
                         ABCD   .DBY $ABCD       REIHENFOLGE HIGH-LOW
```

ALS SPEICHERAUSZUG FINDEN WIR NACH DIESEN ANWEISUNGEN:

<M>=0700 01 03 01 02 03 04 0D 41 42 43 44 CD AB AB CD.

3.7 Anweisungen für die Assemblerliste

Die Anweisung .PAGE erzeugt eine gestrichelte Linie, die sich zur Abhebung von Überschriften eignet. .PAGE 'TITEL', also Anweisung mit nachfolgendem Text in Anführungszeichen erzeugt den Abdruck von TITEL in der auf die Strichelung folgenden Zeile. Weitere Überschriftszeilen können als Kommentare gedruckt werden, wenn dem Text je Zeile ein Semikolon (;) vorangestellt wird. Kommentarzeilen sollten in Quellenprogramme möglichst zahlreich aufgenommen werden. Innerhalb einer Befehls- oder Anweisungszeile können Kommentare mit Zwischenraum auf den Anweisungstext folgen.

.SKIP erzeugt einen Zeilentransport in der Assemblerliste. Die Anweisung kann daher zur Abhebung von Programmsegmenten eingesetzt werden.

Für den Ausdruck gibt es drei mit .OPT...im Quellenprogramm steuernde Anweisungspaare. .OPT LIS/.OPT NOL erzeugt volle Umwandlungsliste bzw. nur die Fehlerliste. .OPT GEN/.OPT NOG erzeugt für alle/für nur 2 ASCII-Zeichen einer .BYTE-Anweisung den Ausdruck dieser Zeichen. .OPT ERR/.OPT NOE erzeugt entweder nur die Meldung von fehlerhaften Zeilen des Quellenprogrammes oder die vollständige Assemblerliste zusammen mit den Fehlermeldungen. Listanweisungen im Quellenprogramm haben zur Zeit der Assemblierung Vorrang vor entsprechenden Anweisungen von der Tastatur her.

4. Zusammenfassung

Zweck dieses Aufsatzes konnte es nur sein, die Ausführungen im Anwenderhandbuch durch Beispiele und durch das Aufzeigen von Zusammenhängen zu ergänzen. Insbesondere waren die Möglichkeiten des Inputs und die verschiedenen Outputformen aufzuzeigen sowie die unterschiedlichen Schreibweisen im Quellenprogramm des Editors und in der Assemblerliste hervorzuheben. Das Handbuch hätte in diesen Abschnitten ausführlicher sein dürfen.

Es ist nun an dem Leser, sich die zahlreichen gegebenen Möglichkeiten der Assemblerprogrammierung durch Übung zu eigen zu machen. Anregungen in großer Zahl findet er dafür nicht nur im Abschnitt 5 des Handbuches, sondern auch an anderen Stellen (Abschnitt 7, 8 und K).

R. L.

AIM-BASIC

Specialities of AIM BASIC are treated in this article, the command set, linking with machine programs by USR and possibilities to SAVE programs.

Seit Mitte Mai werden die z.T. seit langem erwarteten BASIC-ROMs ausgeliefert, zusammen mit einem etwa 90-seitigen BASIC LANGUAGE REFERENCE MANUAL in Englisch. Es handelt sich um ein 8 kBASIC der Firma Microsoft, die bekanntlich Interpreter für zahlreiche Maschinen geschrieben hat. Für den Interessenten, der die BASIC-Option noch nicht bestellt hat und für den Anwender folgende Informationen:

Folgende Anweisungen sind implementiert (S. TABELLE)

Auf die Wirkung dieser Anweisungen und Befehle kann hier nicht im einzelnen eingegangen werden. Der noch wenig vorbereitete Leser sollte sich zur Einarbeitung auf die inzwischen doch recht umfangreiche BASIC-Literatur und natürlich auf die Systemhandbücher stützen. In Deutsch sind u.a. zwei im Oldenbourg-Verlag erschienene Bücher zu empfehlen: J. Schärf und A. Kunesch „BASIC für Kaufleute" und H.O. Ramp „BASIC-Praxis". Mit gutem Gewinn wird man Handbücher von MDT-Computern heranziehen können, so z.B. das „BASIC-Handbuch System 2200" von Wang. In amerikanischen Zeitschriften werden viele Titel inseriert. Hier wurden als empfehlenswert geprüft: „Advanced BASIC - Applications and Problems" von James S. Coan, Verlag Hayden und das viele wissenschaftliche Anwendungen mit Lösungsansätzen abdeckende „Scientific and Engineering Problem-Solving with the Computer" von William Ralph Bennet jun, erschienen bei Prentice Hall, 457 Seiten.

AIM-BASIC hat eine Genauigkeit von 9 Stellen. Der Befehlssatz ist geringer, als z.B. beim PET2001. Beide haben kein PRINTUSING, mit dem man die Ausgabe formatieren kann. Für den AIM fehlen Statements wie OPEN und CLOSE sowie das SYS-Statement, mit dem man unter Angabe der Startadresse in ein Unterprogramm in Maschinensprache eintauchen kann. Nicht implementiert sind ferner die Konstante Pi, die ATN-Funktion (für die aber im Handbuch ein Unterprogramm in Maschinensprache enthalten ist) und das VERIFY für die Prüfung von Dumps auf dem Magnetband.

IMPLEMENTIERT SIND FOLGENDE BEFEHLE UND ANWEISUNGEN

BEFEHLE
- CLEAR
- CONT
- FRE
- LIST
- LOAD
- NEW
- PEEK
- POKE
- RUN
- SAVE

ANWEISUNGEN
- DEF FN
- DIM
- END
- FOR
- GOSUB
- GOTO
- IF...GOTO
- IF...THEN
- LET
- NEXT
- ON...GOSUB
- ON...GOTO
- REM
- RESTORE
- RETURN
- STOP
- USR
- WAIT

EINGABE/AUSGABE
- DATA
- GET
- INPUT
- POS
- PRINT
- READ
- SPC
- TAB

STRING-FUNKTIONEN
- ASC
- CHR$
- LEFT$
- LEN
- MID$
- RIGHT$
- STR$
- VAL

ARITHMETISCHE FUNKTIONEN
- ABS
- ATN *(siehe Text)*
- COS
- EXP
- INT
- LOG
- RND
- SIN
- SGN
- SQR
- TAN

Beim Initialisieren des BASIC mit dem Monitor-Command (5) wird der Benutzer nach der verfügbaren Speichergröße und nach der Zeilenbreite der Ausgabeeinheit gefragt. Damit kann man einen Speicherbereich gegen das Überschreiben durch BASIC schützen und in den vorhandenen oberen Adressen Unterprogramme in Maschinensprache ansiedeln.

Zum Betrieb des BASIC-Interpreters gehört nach Ansicht des Autors eigentlich ein Bildschirmdisplay. BASIC ist eine interaktive Sprache, mit der man Fragen an den Bediener richten und in der man Tabellenbeschriftungen und Ergebnisse formatiert ausgeben kann. Arbeitet man vorwiegend interaktiv mit dem LED-Display, so bedeuten jeweils nur 20 angezeigte

Zeichen doch eine ziemliche Beschränkung, auch wenn die aneinandergereihte Zeilenbreite für Display und Printer 60 Zeichen beträgt. Wir gehen an anderer Stelle auf Fragen des TV-Displays ein. In diesem Zusammenhang wird man sich natürlich die Datenübertragungsgeschwindigkeit ansehen. Bei Anschluß eines Bildschirmterminals, wie hier mit 20 mA-Stromschleife angeschlossen und mit 300 Baud betrieben, benötigt das nachstehende kleine Tabellierprogramm etwa 89 Sekunden. Bei direkter Verbindung des Video-RAMs mit den Prozessorbussen wie etwa im PET benötigt man knapp 4 Sekunden:

```
FOR I = 1 TO 60:PRINTTAB(I);I:NEXT I
```

Der AIM 65 ist ein Gerät, das durch den Monitor und durch die Assembleroption hervorragend für die Programmierung in Maschinensprache ausgerüstet ist. Solche Unterprogramme kann man auch durch ein BASIC-Programm erzeugen, der PET 6502-Assembler in Heft 6 ist ein Beispiel dafür. Die hierfür einzusetzende BASIC-Instruktion heißt POKE I,J. Dabei sind I und J Dezimalzahlen. I gibt die Maschinenadresse an, unter der der Wert J abgelegt werden soll. Der von der Maschinensprache herkommende Leser wird nun den Befehl A9 55 (für LDA #$ 55) sicher für merkfähiger als das Zahlenpaar 169 und 85 halten. Gleichwohl muß er in BASIC wie folgt programmieren, wenn er diesen Befehl an die hexadezimale Adresse 0F00 (dezimal 3840) bringen will:

```
10 DATA 169, 85
20 FOR I=0 TO 1
30 READ A
40 POKE 3840+I,A
50 NEXT I
```

ODER KÜRZER:

```
10 POKE 3840,169
20 POKE 3841,85
```

Für den auf Maschinensprache orientierten AIM wird es bei längeren Programmen nützlicher sein, solchen Objectcode unabhängig vom BASIC-Programm in die hohen Speicherregionen zu laden. Das verlangt natürlich etwas zusätzliche Aufmerksamkeit.

Bei der ersten Prüfung kurz vor Redaktionsschluß konnte keine Anweisung wie PRINT # T, PEEK (I) entdeckt werden, mit der man den Inhalt von Speicherzellen — und damit Maschinenprogramme — als Datafile auf Magnetband absetzen könnte.

Für den Anschluß an die Maschinenprogrammierung folgende weiteren Hinweise: Man kann natürlich auch den Inhalt von Speicherzellen und Ports abfragen, und zwar mit dem Command PRINTPEEK(I), wobei I wiederum der dezimale Wert der Adresse ist. Als Ergebnis erhalten wir ebenfalls eine Dezimalzahl als Ausdruck, z.B. 160 für hex AO oder 255 für hex FF.

Wie bringt man nun ein im Speicher vorhandenes Unterprogramm in Maschinensprache zum Aufruf durch BASIC und zur Ausführung? Wir schon erwähnt, es gibt keinen Command SYS(I), der zu einer Verzweigung an die dezimal kodierte Startadresse I des Unterprogrammes führt (Unterprogramm heißt hier wie immer in der Maschinensprache: Abschluß der Sequenz mit RTS, entspr. Opcode 60).

Dem Benutzer steht für solche Zwecke der Command X=USR(W) zur Verfügung. Der genaue Programmablauf nach USR konnte in Ermangelung eines Listings noch nicht rekonstruiert werden. USR ist eine Funktion, deren berechneten Wert man im aufrufenden BASIC-Programm weiterverwenden kann. Der Wert des Parameters W wird zunächst als Ganzzahl im Gleitkomma-Akkumulator des Interpreters abgelegt, wo er bearbeitet werden könnte. Alsdann erfolgt ein Sprung mit JUMP (indirekt) an die Adresse 0004/05 in das Unterprogramm. Nach dessen Abschluß wird der im Gleitkomma-Akku enthaltene Wert der Variablen X zugewiesen und man kehrt zum BASIC zurück, ob man nun am Gleitkomma-Akku gearbeitet hat oder nicht. Wenn nicht, findet man in X den unveränderten Wert von W.

Dieser Ablauf bedingt, daß man im Vorwege den Adreßvektor des Unterprogrammes an die Adresse 0004/05 gebracht hat. Nehmen wir an, das Unterprogramm soll in Adresse OFOO beginnen. Dann kodieren wir in BASIC:

```
10 DATA 0, 15
20 FOR I=0 TO 1
30 READ A
40 POKE 4+I,A
50 NEXT I
60 X=USR(0)
```

ODER GANZ EINFACH

```
40 POKE 4,0
50 POKE 5,15
60 X=USR(0)
```

Zu dieser Sequenz muß man bedenken, daß der Parameter '0' und auch die Variable 'X' dummies sind, Werte, die auf das Unterprogramm in Maschinensprache keinen Einfluß haben müssen. Der Aufruf von USR erfordert eben die Zuweisung an eine Variable und einen Parameter. Wenn wir noch eine Zeile 70 hinzufügen 70 PRINT X, so finden wir in X den Wert '0', sofern das Unterprogramm den Gleitkomma-Akku nicht veränderte. Der Weg zur Maschinenprogrammierung und von dort aus zurück führt also über Umleitungen. Vielleicht findet man im Laufe der Zeit einen direkteren Einstieg in solche Unterprogramme. Wir können aber auch ohne memory map des BASIC für die Zeropage ablesen, daß man den Gleitkomma-Akku als Arbeitsbereich benutzen kann, solange man sich nicht darum kümmert, was aus dem Wert der Variablen X wird.

Die Dokumentation des AIM-BASIC mit der erwähnten Fibel steht sicher erst am Anfang. Der Leser wird bemerkt haben, daß man den Befehl PRINT mit ? (Fragezeichen) abkürzen kann. Solche Abkürzungen wird es auch für fast alle anderen Commands geben. Sie sind leider noch nicht veröffentlicht, ebenso nicht das für manche Verfeinerungen nützliche memory map. Beim PET hat man etwa ein Jahr auf diese Angaben warten müssen. Es ist zu hoffen, daß Rockwell es schneller schafft.

Bleibt die Frage der Abspeicherung der eingetippten BASIC-Programme. Laut Manual gibt es nur die Abspeicherung über Dump des Editors (unter gewissen Richtlinien) oder das SAVE in BASIC mit OUT=T. Kurze Versuche zeigen aber, daß man mit OUT=D oder P oder X eine Anzeige auf dem Display und dem Drucker erhält, wenn man von der AIM-Tastatur her arbeitet. Kommt man von der Terminal-Tastatur, so ist mit diesen Anweisungen kein Druck auf dem Printer möglich. OUT=L führt zu einer Rückkehr ins BASIC, was auf Lochstreifenstanzung schließen läßt (konnte hier nicht getestet werden). Bleibt da noch die Frage des OUT=U (User defined). Hierzu konnten ebenfalls noch keine abschließenden Erkenntnisse gewonnen werden.

User-Output verlangt einen Adreßvektor in $010A. Wenn man von BASIC aus mit OUT=U über diese indirekte Adresse ein kleines Maschinenprogramm anspringt, so gelangt dieses zwar zur Ausführung, es erfolgt aber keine Rückkehr entweder zum Monitor oder zum BASIC, auch ein eingeschobenes PLA verbessert die Situation nicht. Eine User-Funktion wäre aber ganz wünschenswert, entweder für schnellere Magnetbandformate oder für Floppy-Disk, um einen höheren Datendurchsatz zu ermöglichen.

Ein weiterer Hinweis noch zum Aufruf von Maschinenprogrammen vom BASIC her: Besser noch als die Funktion USR ist die Funktion ATN geeignet. Sie ist bekanntlich nicht im BASIC-ROM implementiert, kann aber im RAM abgebildet werden (siehe Anhang H im BASIC Language Reference Manual). Ihr Aufruf führt zu einem indirekten Sprung zu einem Maschinenprogramm, dessen Vektor in den Zellen 188 und 189 (dezimal) niedergelegt ist. Dabei wird kein Umweg über den Fließkomma-Akku eingeschlagen. Wie im Handbuch beschrieben, legt man den Adreßvektor mittels Befehl POKE 188,.. POKE 189,.. nieder.

R. L.

MOVE AND RELOCATE

Ingo Dohmann, Im Wiehagen 15, D-4830 Gütersloh 12

E: Utilities MOVIT and RELOCATE from »The First Book of KIM« are combined in an interactive AIM 65 program. This programm will either move programs or data to a new destination in memory or will move and recalculate addresses and branches for the new locations.

Im »First Book of KIM« sind zwei sehr nützliche Dienstprogramme abgedruckt, MOVIT von Lew Edwards und RELOCATE vom Jim Butterfield. Mit dem ersten können Programme und Daten beliebig im Speicher verschoben werden, das zweite dient der Editierung von Maschinenkode beim Einfügen oder Fortlassen von Instruktionen. Es führt notwendige Umrechnungen von Einsprungs-, Verzweigungs-, Daten- und Arbeitsbereichsadressen aus. Beide Programme standen bisher unabhängig nebeneinander und waren etwas umständlich manuell zu bedienen. Im RELOCATE bildeten insbesondere Tabellen einen Fremdkörper.

Es ist Herr Dohmann zu danken, beide Programme vereint zu haben, in dem er ihnen ein interaktives AIM-Programm überlagerte. In dieser Synthese werden die Arbeitsbereiche der ursprünglichen Programme noch beibehalten, es findet dabei ein Parametertransport vom Steuerprogramm und vom Programmteil MOVIT her statt. Man könnte sicher noch gewisse Kürzungen vornehmen. Angesichts der vielen damit verbundenen Mühe wird das nur selten lohnen.

Das Programm ist auf die obersten Speicherstellen eines 4 k-AIM gelegt worden. In vielen Fällen wird es einen Assembler entbehrlich machen, denn die Leistungsfähigkeit der Instruktionseingabe unter I-Command ist schon beachtlich und wird durch diese utility noch gesteigert. Der Nachdruck der zitierten und weiter nicht kommentierten Originalprogramme erfolgte mit freundlicher Genehmigung des europäischen Copyright-Inhabers.

Zur Programmbedienung: Man lade den Sprungbefehl 4C 7C OD für eine der Funktionstasten F1, F2 oder F3 in Adresse 010C ff. Das Programm meldet sich mit 'Verschieb und Umrechnen' und 'FROM= und To='. Hier trägt man den zu transportierenden Speicherbereich ein. Die Abfrage 'New Begin=' betrifft das Ziel des Transports. Die Abfrage 'Relocat?' führt mit 'N' zum sofortigen Datentransport ohne Abänderungen. Bei 'Y' wird der Programmanfang abgefragt, danach das Programmende (INCLUSIVE TABLE, die von der Umrechnung ausgeschlossen wird). Schließlich wird man nach der auf die letzte Instruktion folgenden Adresse gefragt (das kann ggfs. der erste Tabellenplatz sein). RETURN löst Transport und Umrechnung aus. Das Programm kehrt zum Monitor zurück.

```
D0 = OSAL          STARTADRESSE, ALT
D1 = OSAH
D2 = OEAL          ENDADRESSE, ALT
D3 = OEAH
D4 = NSAL          STARTADRESSE NEU
D5 = NSAH
D6 = NEAL          ENDADRESSE NEU
D7 = NEAH
D8 = BCL           BYTE COUNT
D9 = BCH

010F 4C JMP 0D7C

0D7C 20 JSR 0DFD   START
  7F 4C JMP E182   ENDE
  82 E6 INC D8
  84 E6 INC D9
  86 A5 LDA D6
  88 85 STA EA
  8A A5 LDA D7
  8C 85 STA EB
  8E 60 RTS
```

```
0D8F EA NOP
  90 D8 CLD          ENTSP. MOVIT
  91 A0 LDY #FF
  93 38 SEC
  94 A5 LDA D2
  96 E5 SBC D0
  98 85 STA D8
  9A A5 LDA D3
  9C E5 SBC D1
  9E 85 STA D9
  A0 18 CLC
  A1 A5 LDA D8
  A3 65 ADC D4
  A5 85 STA D6
  A7 A5 LDA D9
  A9 65 ADC D5
  AB 85 STA D7
  AD 20 JSR 0D 82    MODIFIZIERT
  B0 EA NOP
  B1 38 SEC
```

```
   B2 A5 LDA D4
   B4 E5 SBC D0
   B6 A5 LDA D5
   B8 E5 SBC D1
   BA A2 LDX #00
   BC 90 BCC 0DC0
   BE A2 LDX #02
   C0 A1 LDA (D0,X)
   C2 81 STA (D4,X)
   C4 90 BCC 0DDA
   C6 C6 DEC D2
   C8 98 TYA
   C9 45 EOR D2
   CB D0 BNE 0DCF
   CD C6 DEC D3
   CF C6 DEC D6
   D1 98 TYA
   D2 45 EOR D6
   D4 D0 BNE 0DD8
   D6 C6 DEC D7
   D8 B0 BCS 0DE6
   DA E6 INC D0
   DC D0 BNE 0DE0

0DDE E6 INC D1
   E0 E6 INC D4
   E2 D0 BNE 0DE6
   E4 E6 INC D5
   E6 C6 DEC D8
   E8 D0 BNE 0DEC
   EA C6 DEC D9
   EC D0 BNE 0DBA
   EE 60 RTS

0DEF B9 LDA 0F90,Y  TAB IN AKKU
   F2 48 PHA        AKKU SICHERN
   F3 29 AND #7F    OHNE BIT 7
   F5 20 JSR E97A   OUTPUT
   F8 C8 INY
   F9 68 PLA        AKKU ZUR.
   FA 10 BPL 0DEF   BIT 7 ≠ 1
   FC 60 RTS

0DFD 20 JSR EA13    HAUPTPROGR.
0E00 A0 LDY #00     NEUE ZEILE, TEXT
   02 20 JSR 0DEF   AUSGABE TABELLE
   05 20 JSR EA13   NEUE ZEILE
   08 20 JSR E7A3   FROM=
   0B B0 BCS 0E08   BEI FEHLER
   0D 20 JSR E83E   ZWISCHENRAUM
   10 AD LDA A41C   ADDR, ALTEN BEGINN
   13 AE LDX A41D   ADDR+1
   16 85 STA D0     FÜR MOVIT
   18 86 STX D1
   1A 20 JSR E7A7   TO=
   1D B0 BCS 0E1A   BEI FEHLER
   1F AD LDA A41C   ADDR, ALTES ENDE
   22 AE LDX A41D   ADDR+1
   25 85 STA D2     FÜR MOVIT
   27 86 STX D3
   29 20 JSR EA13   NEUE ZEILE
   2C A0 LDY #14    ANF (NEW BEGIN)
   2E 20 JSR 0DEF   AUSG. TABELLE
   31 20 JSR EAB1   EINGABE ADRESSE
   34 B0 BCS 0E2C   NOCHMAL BEI FEHLER
   36 AD LDA A41C   NEUEN ANFANG
   39 AE LDX A41D   IN DEN ARBEITS-
   3C 85 STA D4     SPEICHER VON
   3E 86 STX D5     VON MOVIT
   40 20 JSR EA13   NEUE ZEILE
   43 A0 LDY #1F    ANFANG DER RELOC?
   45 20 JSR 0DEF   AUSGABE TABELLE
   48 20 JSR E973   EINGABE Y ODER N
   4B C9 CMP #59    VERGLEICH "Y"
   4D D0 BNE 0E95   UNGLEICH

0E4F 20 JSR EA13    NEUE ZEILE
   52 A0 LDY #27    ANF (PROGR. START)
   54 20 JSR 0DEF   AUSGABE TEXT
   57 20 JSR EAB1   EINGABE ADRESSE
   5A B0 BCS 0E52   BEI FEHLER
   5C AD LDA A41C   PROGRAMM-START-
   5F AE LDX A41D   ADRESSE SICHERN
   62 85 STA D8
   64 86 STX D9
   66 20 JSR EA13   NEUE ZEILE
   69 20 JSR 0EB3   RUCKSACK
   6C EA NOP
   6D 20 JSR 0ED7   ENDE MIT TABELLE
   70 A0 LDY #32    HOLEN. ANF?
   72 20 JSR 0DEF   TEXT
   75 20 JSR EAB1   EINGABE ADRESSE
   78 B0 BCS 0E70   BEI FEHLER
   7A AD LDA A41C   ADRESSE FÜR INDI-
   7D AE LDX A41D   REKTE ADRESSIERUNG
   80 85 STA D6     ZWISCHENSPEICHERN
   82 86 STX D7
   84 A0 LDY #00    FF HINTER LETZTE
   8  B1 LDA (D6),Y INSTRUKTION. RETTE
   88 85 STA 00
   8A A9 LDA #FF
   8C 91 STA (D6),Y
   8E 20 JSR 0EE3   START F. RELOC.
   91 A5 LDA 00     CHAR. ZURÜCK
   93 91 STA (D6),Y
   95 20 JSR 0D90   MOVIT
   98 20 JSR EA13   NEUE ZEILE
   9B A0 LDY #52    END
   9D 20 JSR 0DEF   TEXTAUSGABE
   A0 A5 LDA EA     NEUES BLOCKENDE
   A2 A6 LDX EB     AUSGEBEN
   A4 8D STA A425   (DISASSEMBLER)
   A7 8E STX A426
   AA A9 LDA #01
   AC 8D STA A419
   AF 20 JSR E720
   B2 60 RTS
```

```
B3 A0 LDY #55     ENDE INCL. TABELLE
B5 20 JSR 0DEF    TEXT
B8 20 JSR EAB1    EINGABE ADRESSE
BB B0 BCS 0EB3    BEI FEHLER
BD AD LDA A41C    ENDE IN ARBEITS-
C0 AE LDX A41D    SPEICHER VON RELOC.
C3 85 STA E6
C5 86 STX E7
C7 60 RTS
C8 D8 CLD         VERSCHIEBEDIS-
C9 38 SEC         TANZ BERECHNEN
CA A5 LDA D4      UND IM ARBEITS-
CC E5 SBC D0      SPEICHER VON
CE 85 STA E8      RELOC. ABLEGEN

0ED0 A5 LDA D5
D2 E5 SBC D1
D4 85 STA E9
D6 60 RTS
D7 A5 LDA D0      BEGINN DER
D9 A6 LDX D1      VERSCHIEBUNG
DB 85 STA EC      VON ARBEITSSPEICHER
DD 86 STX ED      MOVIT N. RELOC.
DF 20 JSR 0EC8    DISTANZ BERECHNEN
E2 60 RTS

E3 A5 LDA D8      DITO EINSETZEN
E5 A6 LDX D9      PROGRAMM-START
E7 85 STA EA
E9 86 STX EB
EB 20 JSR 0EF1    RELOCATE !
EE A0 LDY #00
F0 60 RTS

0EF1 D8 CLD       PROGRAMMFOLGE
F2 A0 LDY #00     AUS RELOCATE
F4 B1 LDA (EA),Y
F6 A8 TAY
F7 A2 LDX #07
F9 98 TYA
FA 3D AND 0F79,X
FD 5D EOR 0F80,X
0F00 F0 BEQ 0F05
02 CA DEX
03 D0 BNE 0EF9
05 BC LDY 0F88,X
08 30 BMI 0F17
0A F0 BEQ 0F2E
0C E6 INC EA
0E D0 BNE 0F12
10 E6 INC EB
12 88 DEY
13 D0 BNE 0F0C
15 F0 BEQ 0EF1
17 C8 INY
18 30 BMI 0F79
1A C8 INY
1B B1 LDA (EA),Y
1D AA TAX
1E C9 INY
1F B1 LDA (EA),Y
21 20 JSR 0F5A
24 91 STA (EA),Y
26 88 DEY
27 8A TXA
28 91 STA (EA),Y
2A A0 LDY #03
2C 10 BPL 0F0C
2E C8 INY
2F A6 LDX EA
31 A5 LDA EB
33 20 JSR 0F5A

0F36 86 STX E0
38 A2 LDX #FF
3A B1 LDA (EA),Y
3C 18 CLC
3D 69 ADC #02
3F 30 BMI 0F42
41 E8 INX
42 86 STX E3
44 18 CLC
45 65 ADC EA
47 AA TAX
48 A5 LDA E3
4A 65 ADC EB
4C 20 JSR 0F5A
4F CA DEX
50 CA DEX
51 8A TXA
52 38 SEC
53 E5 SBC E0
55 91 STA (EA),Y
57 C8 INY
58 10 BPL 0F0C
5A C5 CMP E7
5C 90 BCC 0F68
5E D0 BNE 0F79
60 E4 CPX E6
62 90 BCC 0F68
64 F0 BEQ 0F68
66 D0 BNE 0F79
68 C5 CMP ED
6A D0 BNE 0F6E
6C E4 CPX EC
6E 90 BCC 0F79
70 48 PHA
71 8A TXA
72 18 CLC
73 65 ADC E8
75 AA TAX
76 68 PLA
77 65 ADC E9
79 60 RTS
```

TABELLE FÜR RELOCATION

```
<M>=0F7A 0C 1F 0D 87
< > 0F7E 1F FF 03 0C
< > 0F82 19 08 00 10
< > 0F86 20 03 02 FF
< > 0F8A FF 01 01 00
<>  0F8E FF FE
```

TABELLE FÜR TEXT

```
<M>=0F90 56 45 52 53
< > 0F94 43 48 49 45
< > 0F98 42 2E 26 55
< > 0F9C 4D 52 45 43
< > 0FA0 48 4E 45 CE
< > 0FA4 4E 45 57 20

<M>=0FA8 42 45 47 49
< > 0FAC 4E 4E BD 52
< > 0FB0 45 4C 4F 43
< > 0FB4 41 54 BF 50
< > 0FB8 52 4F 47 2E
< > 0FBC 53 54 41 52
< > 0FC0 54 BD 41 44
< > 0FC4 44 52 45 53
< > 0FC8 53 20 42 45
< > 0FCC 48 49 4E 44
< > 0FD0 20 4C 41 53
< > 0FD4 54 0D 49 4E
< > 0FD8 53 54 52 55
< > 0FDC 43 54 49 4F
< > 0FE0 4E BD 45 4E
< > 0FE4 C4 45 4E 44
< > 0FE8 20 49 4E 43
< > 0FEC 2E 54 41 42
< > 0FF0 4C 45 20 BD
< > 0FF4 2E 26 55 4D
< > 0FF8 52 45 43 48
< > 0FFC 4E 45 CE 4E
```


****

MEHRERE IBM-KK-SCHREIBMASCHINEN ALS I/O-MASCHINEN MODIFIZIERT (Ansteuermagnete und Abfragekontakte sind eingebaut und verdrahtet) völlig überholt und getestet, privat zu verkaufen. Ein Schaltungsvorschlag als TTY-Ersatz wird mitgeliefert.

Preis: Typ 723/DM 1450,- und Typ 725/DM 1500,-.
Gerrit Krause, An der Schmitte 15, 5657 Haan 1
Tel. 02129 - 7872, ab 19 Uhr.

****

KLEINANZEIGEN DER LESER

Verkaufe KIM-1 mit deutscher Beschreibung für DM 420,-. W. D. Hein, Kollenrodtstr. 17, 3000 Hannover 1, Tel. 0511 - 62 80 80.

Datenaustausch zwischen KIM oder anderen 65xx-Systemen und dem Mikrocomputersystem TRS-80

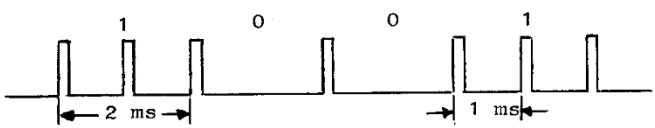
Von Dr. Claus Wünsche, Eschenweg 2, 5778 Meschede

E: A simple method is shown to exchange data between microcomputer systems. The audio tape format of the TRS-80 computer with 500 Baud has been realized with KIM-1. No other interface is needed than a simple IN/OUT port. Programs for KIM-1 are given to read a TRS-80 tape or to generate a TRS-80 readable tape.

In diesem Beitrag wird eine einfache Möglichkeit zum Datenaustausch zwischen Mikrocomputersystemen aufgezeigt. Das Kassettenaufzeichnungsformat des TRS-80 mit einer Übertragungsgeschwindigkeit von 500 Baud kann mit einem Mikroprozessor ohne Hardwareergänzung an einem IN/OUT-Port zum Lesen oder Schreiben realisiert werden. Es werden Programme für den KIM zum Einlesen eines auf dem TRS-80 erzeugten String-File und zur Gewinnung einer TRS-80 kompatiblen Kassettenaufnahme mit dem KIM mitgeteilt. Diese Möglichkeit des Datenaustausches wird für folgende Anwendungen verwendet:

1.) Steuerung einer automatischen Meßvorrichtung mit dem KIM. Die gewonnenen Daten werden im KIM gespeichert und blockweise auf Band abgelegt. Die Weiterverarbeitung der Daten erfolgt im TRS-80.
2.) Darstellung eines Speicherauszuges des KIM auf dem Bildschirm des TRS-80
3.) Verwendung des in Nr. 6 dieser Zeitschrift veröffentlichten in BASIC geschriebenen Assemblers für 65xx Prozessoren. Mit geringen Änderungen wurde er für den TRS-80 angepaßt. Der erzeugte Maschinencode wird per Band in den KIM übertragen.

Das Kassettenformat des TRS-80 arbeitet mit einem Impulsraster der Periode 500 Hz zur Synchronisation. Zwischen diese Impulse wird für eine '1' ein zusätzlicher Impuls eingeschoben (Impulsabstand jetzt 1 ms), für eine '0' bleibt der Impulsabstand 2 ms.



Die Impulslänge beträgt ca. 100 µs. Bedingt durch das schlechte Impulsverhalten der Kassettengeräte wird die Impulsform verzerrt. In dem vorliegenden Programm wird durch den KIM ein 88 µs langer Impuls erzeugt. Damit werden einwandfrei lesbare Bänder erhalten. Die Amplitude muß allerdings sorgfältig eingestellt werden, und gutes Bandmaterial muß verwendet werden.
Eine Bandaufnahme beginnt mit einem Vorlauf von ca. 800 Impulsen mit 2 ms Abstand. Die Daten werden im ASCII-Code ohne Paritätsbit mit MSB zuerst ausgegeben. Als Startzeichen dient % + Paritätsbit, d.h. A5 (Hex). Die Trennung zweier BASIC-Variablen erfolgt durch " ' ", d.h. als Trennzeichen dient 2C (Hex). Das Ende einer Aufnahme wird durch CR (carriage return), d.h. OD (Hex) angezeigt.
Damit die Lage des Bit-Impulses zwischen den Sychronisationsimpulsen nicht kritisch ist, wird durch KIM beginnend 895 µs nach dem Synchronisationsimpuls insgesamt 16 mal auf Bit =1?, d.h. über einen Zeitraum von 208µs abgefragt. Wenn innerhalb dieses Zeitraums keine 1 gefunden wurde, wird Bit ="0" gesetzt. Auf diese Weise wird Impulsverzerrungen durch den Kassettenrecorder Rechnung getragen.

Das Programm zur Erzeugung einer TRS-80 kompatiblen Kassettenaufnahme wird durch die Unterprogramme WORT und BYTE auf die gewünschte Variable in BASIC angepaßt. Hier wird nur die Ausgabe einer String-Variable aus Hexadezimalzeichen beschrieben, wofür WORT und BYTE identisch sind. Mit etwas größerem Aufwand können auch Gleitkommavarlable mit einfacher oder doppelter Genauigkeit oder Integer-Variable ausgegeben werden. Einzelheiten zur Übertragung von Gleitkommazahlen können vom Autor auf Wunsch mitgeteilt werden. Das Programm zum Einlesen einer Kassettenaufnahme des TRS-80 durch den KIM wurde nur für Hexadezimalzeichen niedergeschrieben, da bisher nur hierfür Bedarf besteht.

Die Programme starten in OCOO bzw. ODOO. Zur Verschiebung in andere Speicherbereiche müssen die Adressen der Unterprogramme in den Aufrufen geändert werden. Absolute Sprünge erfolgen sonst nur am Programmende zurück in das Monitorprogramm. In der Programmniederschrift wurde der Einfachheit halber auf das $-Zeichen bei der Angabe von HEX-Adressen verzichtet. Die Startadresse des ein- oder auszugebenden Speicherbereichs im KIM wird fortlaufend erhöht, so daß durch wiederholte Starts aufeinanderfolgende Speicherbereiche eingelesen oder ausgegeben werden können.

Programm zum Einlesen eines String-File des TRS-80 in KIM
(1 String-Variable)
Das Programm ist für eine gerade Anzahl von HEX-Zeichen niedergeschrieben, die im KIM byteweise gepackt werden. Sollen beliebige im ASCII-Code übertragene Zeichen unverändert abgespeichert werden, so wird ab Adresse OD28 ein Sprung (4C 4O OD) eingesetzt. Die Wandlung in HEX-Zeichen und das Packen werden dadurch übersprungen. Für Wandlung und Packen wird die KIM-Routine PACKT (ab 1AOO) verwendet. Die Anfangsadresse in 17F5 und 17F6 wird hochgezählt. Dadurch kann zum Einlesen aufeinanderfolgender Zeichenketten sofort wieder bei Adresse ODOO gestartet werden. Das Programm endet bei Erreichen der Endadresse oder eines Endzeichens CR und der Monitor meldet sich mit der letzten Adresse +1. Kann ein Zeichen nicht als HEX-Zeichen interpretiert werden ,so meldet sich der Monitor mit FFFF.

```
<17F5>,<17F6> ADBE  Startadresse für Zeichenkette
<17F7>,<17F8> ADEN  Endadresse+1
<17E9>        SAVX
<00F7>        BYTE  momentane Daten
<00FA>,<00FB> ADMO  momentane Adresse
 X und Y werden verwendet
 PB2    Eingang  Signal
 PB1    Ausgang  Band Start/Stop
```

```
ODOO    A9 02          LDA #02       Einrichten Ports
        8D 03 17       STA  1703
        A9 00          LDA #00
        8D 02 17       STA  1702     PB1 low, Band Start
        AD F5 17       LDA  ADBE     Startadresse umspeichern
        85 FA          STA  ADMO
        AD F6 17       LDA  ADBE+1
        85 FB          STA  ADMO+1
OD14    20 70 OD STZ   JSR  HOLBIT   Warten auf Startzeichen
        26 F7          ROL  BYTE     Bit einschieben
        A9 A5          LDA #A5       Lade Startzeichen
        C5 F7          CMP  BYTE     Vergleiche
        DO F5          BNE  STZ
OD1F    20 91 OD ZEICH JSR  HOLBY    HEX-Zeichen 1
        A5 F7          LDA  BYTE
```

```
        C9 0D          CMP #0D        Vergleiche mit Endzeichen
        F0 38          BEQ  ZUEND
0D28    20 00 1A       JSR  PACKT     PACKT in KIM
        98             TYA            Y ≠ 0 kein HEX-Zeichen
        D0 3A          BNE  FEHL
0D2E    20 91 0D       JSR  HOLBY     HEX-Zeichen 2
        A5 F7          LDA  BYTE
        C9 0D          CMP #0D        Vergleiche mit Endzeichen
        F0 29          BEQ  ZUEND
0D37    20 00 1A       JSR  PACKT     PACKT in KIM
        98             TYA            Y ≠ 0 kein HEX-Zeichen
        D0 2B          BNE  FEHL
        AD E9 17       LDA  SAVX
0D40    91 FA          STA (ADMO),Y   HEX-Byte abspeichern
        E6 FA          INC  ADMO      Adresse erhöhen
        A5 FA          LDA  ADMO
        8D F5 17       STA  ADBE
        D0 07          BNE  NUEB
0D4B    E6 FB          INC  ADMO+1

0D4D    A5 FB          LDA  ADMO+1
        8D F6 17       STA  ADBE+1
        A5 FB    NUEB  LDA  ADMO+1    Mit Endadresse ver-
        CD F8 17       CMP  ADEN+1    gleichen
        D0 C6          BNE  ZEICH
        A5 FA          LDA  ADMO
        CD F7 17       CMP  ADEN
        D0 BF          BNE  ZEICH
0D60    A9 02    ZUEND LDA #02
        8D 02 17       STA  1702      PB1 high, Band Stop
        4C 4F 1C       JMP  1C4F      zurück Monitor
        A9 02    FEHL  LDA #02
        8D 02 17       STA  1702      PB1 high, Band Stop
        4C 29 19       JMP  1929      zurück Monitor mit FFFF

0D70    AD 02 17 HOLBIT LDA  1702     Laden PB
        29 04          AND #04        abtrennen Bit 2
        F0 F9          BEQ  HOLBIT    Synchronisationsimpuls ?
0D77    A2 B3          LDX #B3        895 µs warten
        CA       S2    DEX
        D0 FD          BNE  S2
        A2 10          LDX #10
        AD 02 17 S3    LDA  1702      16 mal auf Bit prüfen
        29 04          AND #04          (208 µs)
        D0 05          BNE  S4        Sprung bei Bit = 1
        CA             DEX
        D0 F6          BNE  S3
        18             CLC            Bit = 0
        60             RTS            Bit in Carry
        A2 30    S4    LDX #30        Verzögerung bei   = 1
        CA       S6    DEX              (240 µs)
        D0 FD          BNE  S6
        38             SEC            Bit = 1
0D90    60             RTS            Bit in Carry
```

```
OD91    A0 08     HOLBY LDY #08
        20 70 0D  S5    JSR  HOLBIT    Bit in Carry
        26 F7           ROL  BYTE
        88              DEY
        D0 F8           BNE  S5
OD9B    60              RTS
```

Programm zur Erzeugung einer TRS-80 kompatiblen Kassettenaufnahme mit KIM
Das Hauptprogramm liefert die Impulse, setzt das Startzeichen und verzweigt bei Startzeichen, Trennzeichen oder Endzeichen auf entsprechende Programmabschnitte.
Die Erzeugung des gewünschten Datenformats, das Zählen der Zeichen und das Einfügen des Endzeichens (CR) erfolgt in den Unterprogrammen BYTE und WORT.

```
PB0     Ausgang  Signal
PB1     Ausgang  Band Start/Stop
X und Y werden verwendet

0C00    A9 03            LDA #03
        8D 03 17         STA  1703      Ports einrichten
        A9 01            LDA #01
        8D 02 17         STA  1702      PB1 low, Band Start
0C0A    20 96 0C         JSR  VERZ      Verzögerung 262 ms
        A0 04            LDY #04        Vorlauf 800 Impulse
        A2 C8     L1     LDX #C8
        A9 FA     L2     LDA #FA
        8D 45 17         STA  1745      Timer 2 auf 2 ms
0C16    20 7B 0C         JSR  IMP       Impuls
        AD 47 17  W3     LDA  1747      Warten Timer 2
        10 FB            BPL  W3
        CA               DEX
        D0 F0            BNE  L2
        88               DEY
        D0 EB            BNE  L1        Ende Vorlauf, Y = 0
        84 E9            STY  BYZ       Bytezähler Null setzen
        A9 FA            LDA #FA
        8D 45 17         STA  1745      Timer 2 auf 2 ms
0C2B    A9 6B            LDA #6B
        8D 05 17         STA  1705      Timer 1 auf 856 µs
        A9 A5            LDA #A5        Startzeichen laden
0C32    A2 08     SBYTE  LDX #08        Laden Bitzähler
        2A        BIT    ROL  A         Bit in Carry
        48               PHA            Akku retten
        2C 07 17  W1     BIT  1707      Warten Timer 1
        10 FB            BPL  W1
        90 03            BCC  W2        Sprung bei Bit = 0
0C3D    20 7B 0C         JSR  IMP       Impuls
        2C 47 17  W2     BIT  1747      Warten Timer 2
        10 FB            BPL  W2
0C45    20 7B 0C         JSR  IMP       Impuls
        A9 FA            LDA #FA
        8D 45 17         STA  1745      Laden Timer 2 2 ms
0C4D    A9 6B            LDA #6B
        8D 05 17         STA  1705      Laden Timer 1 856 µs
        68               PLA            Akku holen
        CA               DEX
        D0 DE            BNE  BIT       Ende Schleife Bit
        2A               ROL  A         Byte in Akku wiederher-
```

```
                C9 0D          CMP #0D        Endzeichen ? stellen
                F0 12          BEQ L4
                C9 2C          CMP #2C        Trennzeichen ?
                F0 09          BEQ L3
                C9 A5          CMP #A5        Startzeichen ?
                F0 05          BEQ L3
0C63            20 A1 0C       JSR BYTE       neues Byte
                D0 CA          BNE SBYTE

0C68            20 A1 0C L3    JSR WORT       neues Wort beginnen
                D0 C5          BNE SBYTE
0C6D            20 96 0C L4    JSR VERZ       Verzögerung 262 ms
                A9 02          LDA #02
                0D 02 17       ORA 1702
                8D 02 17       STA 1702       PB1 high, Band Stop
0C78            4C 4F 1C       JMP 1C4F       zurück Monitor

0C7B            A9 01     IMP  LDA #01
                0D 02 17       ORA 1702
                8D 02 17       STA 1702       Signal high
0C83            A9 0B          LDA #0B        Laden Impulslänge
                8D 05 17       STA 1705       Timer 1 x 8/us
                AD 07 17 W4    LDA 1707       Warten Timer 1
                10 FB          BPL W4
                A9 FE          LDA #FE
                2D 02 17       AND 1702
                8D 02 17       STA 1702       Signal low
                60             RTS

0C96            A9 FF     VERZ LDA #FF
                8D 07 17       STA 1707       Laden Timer 1
                AD 07 17 W5    LDA 1707       Warten Timer 1
                10 FB          BPL W5
0CA0            60             RTS
```

Unterprogramm (BYTE / WORT) für eine String-Variable (HEX-Kette)

Folgende Adressen sind vor dem Start zu laden, bzw. werden belegt:

```
<00DC> BYE  Anzahl der auszugebenden Byte (max. 7C für TRS-80 )
<00DD> = 0  HZ  Hilfszähler für Halbbyte, zu Beginn Null
<00DE>, <00DF> SADL,SADH  Startadresse für Zeichenkette
<00E9>      BYZ  Bytezähler
```

```
0CA1    A5 E9     BYTE  LDA  BYZ
        C5 DC           CMP  BYE       Vergleiche auf Ende
        D0 04           BNE  B1
        A9 0D           LDA #0D        Lade Endzeichen CR
        D0 26           BNE  ENDE
        A5 DD     B1    LDA  HZ        Welches Halbbyte ?
        D0 0A           BNE  B2
        B1 DE           LDA (SADL),Y   Laden Byte
        4A              LSR
        4A              LSR
        4A              LSR
        4A              LSR            Bereitstellen Bit 4-7
```

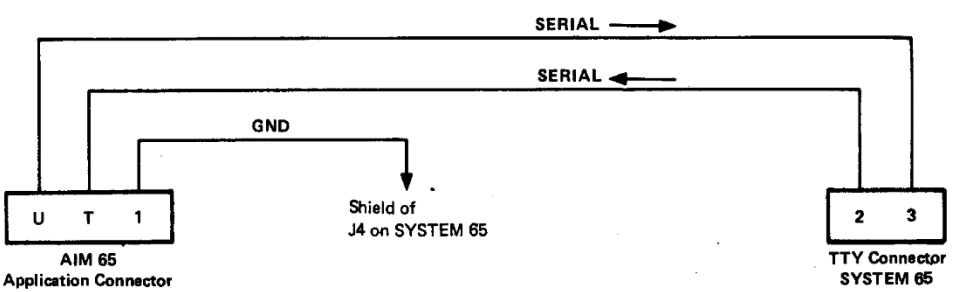
```
                E6 DD          INC  HZ          Hilfszähler auf 1
                D0 0E          BNE  WD
                B1 DE     B2   LDA  (SADL),Y   Laden Byte
                29 0F          AND  #0F         Bereitstellen Bit 0-3
                E6 DE          INC  SADL        Adressen erhöhen
                D0 02          BNE  B3
                E6 DF          INC  SADH
                E6 E9     B3   INC  BYZ         Erhöhe Bytezähler
                C6 DD          DEC  HZ          Hilfszähler auf 0
                D8        WD   CLD              Wandle HEX-Zeichen
                C9 0A          CMP  #0A
                18             CLC
                30 02          BMI  WD1
                69 07          ADC  #07
                69 30     WD1  ADC  #30         Zeichen im ASCII-Code
06D1            60        ENDE RTS
```

****

AIM 65 ALS TERMINAL

Dem Rockwell-Datenblatt 'System 65 to AIM 65 Interface' ist ein Schaltungsvorschlag gemäß nachfolgender Zeichnung zu entnehmen, nach dem AIM 65 als Empfänger von Daten betrieben wird, die vom System 65 ausgehen. Beide Geräte haben bekanntlich den seriellen TTY-Port. Dabei ist am sendenden Prozessor die Zahl der Stop-Bits auf 2 einzustellen, die Baudrate am AIM in den Zellen A417/18 ist gemäß Anwenderhandbuch, Abschnitt 9.2.3 vorzuhalten. Die Datenaussendung kann beginnen, sobald am AIM der Monitor LOAD-Command gegeben wurde mit (L) IN=L.

Dieser Grundgedanke läßt sich für die Zusammenschaltung auch anderer Prozesooren mit dem AIM verwenden. Bei Übertragungsraten oberhalb 110 Baud wird die Entfernung des Kondensators C7 am TTY-Port empfohlen. Man beachte das Datenformat gemäß Anhang H im Anwenderhandbuch.



SYSTEM 65 to AIM 65 Serial Interface

AIM 65 Spezial (2)

Seit Mitte Mai werden die ersten BASIC-ROMs ausgeliefert. Die bisher verfügbaren Mengen reichten zur sofortigen Abdeckung der Vorbestellungen noch nicht aus, so daß hier noch etwas Geduld empfohlen werden muß.

Für das englische Handbuch sind umfangreiche Nachträge/Berichtigungen (sicher über 100) eingegangen. Sie sollen alle in die deutsche Übersetzung des Anwenderhandbuches eingearbeitet sein, die damit einen aktuelleren und zuverlässigeren Stand repräsentieren dürfte, als die englische Vorlage. Wegen dieser Bearbeitung hat sich die Fertigstellung beim Drucker der Firma Rockwell verzögert. Nach den jetzt vorliegenden Zusagen kann die Auslieferung der von den Lesern bestellten Exemplare erst etwa Ende Juni 1979 erfolgen.

Über die Benutzung des AIM-Assemblers wird an anderer Stelle dieses Heftes berichtet, auch das BASIC wird vorgestellt. Als interarktive Programmiersprache bedarf BASIC konsequenterweise der Bildschirmausgabe, denn Übersicht und Wirkung sind auf dem 20-stelligen Display und auf dem Printer zu gering (eine Programmzeile mit 60 Zeichen wird dort als drei willkürlich getrennte Zeilen von je 20 Zeichen dargestellt). Dem Vernehmen nach wird man wahrscheinlich schon im Juni speziell auf den AIM angepaßte Video-Platinen erhalten können. Fest angekündigt wurden sie von I. Dohmann, Elektronische Baugruppen, in Gütersloh und vom Micro-Shop-Bodensee (Nedela), siehe Anzeigenteil.

*

Beim Herausgeber wird seit einiger Zeit die XITEX-Platine SCT-100 zusammen mit einer Encoder-Tastatur von GRI betrieben. Sie ist für die 20 mA- und 300 Baud-schleife des AIM problemlos vorbereitet, stellt 128 Zeichen dar, darunter etliche Sonderzeichen und griechische Buchstaben, ferner natürlich Kleinschreibung. Für viele Fälle, auch in BASIC, ist die Übertragungsrate zu langsam. Nach dem Stand der Technik sollte man bei Neuanschaffungen darauf Wert legen, daß das Video-Ram von der CPU über Adreß- und Datenbus direkt beschrieben werden kann und daß die Erweiterung auch ein Betriebssystem für die Verwaltung des Verkehres zwischen Prozessor und Video-Platine trägt. Dazu wird auch eine Initialisierungsroutine gehören.

Man muß nicht unbedingt einen breitbandigen Bildschirm-Monitor betreiben, ganz passabel geht es auch mit einem normalen Heimfernseher mit Hochfrequenzeinspeisung, wobei die bekannten Schaltungen durchaus unterschiedliche Wiedergabequalität haben. Auch für den Umbau von TV-Geräten mit einer Direkteinspeisung des Bildsignales hinter die ZF eines Fernsehers gibt es zahlreiche Vorschläge, insbesondere auch in den bekannten Büchern von Don Lancaster. Stets wird jedoch vor 'heißen' Chassis gewarnt, Fernsehern ohne Netztrafo. Angesichts geringer TV-Preise in den Versand- und Warenhäusern (unter DM 200,-) sollte man einen sicheren und nicht zu arbeitsaufwendigen Weg wählen.

Der Monitor des AIM unterstützt nicht nur im TTY-Betrieb (Schiebeschalter) ein selbständiges Bildschirmterminal mit separater Tastatur, er läßt auch eine Betriebsweise zu, bei der die AIM-Tastatur aktiv bliebt. Der Umschalter bleibt also auf Keyboard eingestellt, gleichwohl werden alle Betätigungen und Ausgaben auf dem Bildschirm so dargestellt, wie man es von Display und Drucker her gewohnt ist. Der Bildschirm benötigt dann also keine eigene Tastatur, so daß preiswerte Aufbauten möglich sind. Herr Peter Steinkamp in Hamburg hat dafür auf das DI-Link in Adresse A406/07 aufmerksam gemacht. Der AIM-Monitor erzeugt bei Abschluß einer Zeile nur das 'CR'-Signal, nicht jedoch das Line Feed, das vom Bildschirm gefordert wird, damit der nicht immer in die gleiche Zeile schreibt. Er baut daher ein kleines Interims-

programm auf, das zu jedem 'CR' ein 'LF' auf den Bildschirm nachschießt. Außerdem wird in der Initialisierung der DI-Link-Vektro auf diese kleine Routine gerichtet. Hier das Coding, das zwar mittendrin sitzt aber leicht auf andere Speicherplätze geändert werden kann.

```
09A0 8D STA 09CC    AKKU SICHERN              CR-PRÜFUNG
09A3 29 AND #7F     PARITY BIT AUSBLENDEN
09A5 C9 CMP #0D     EIN 'CR'?
09A7 F0 BEQ 09AC    VERZWEIGEN, WENN JA
09A9 4C JMP EEA8    NORMAL NACH OUTTTY
09AC A9 LDA #0A     ASCII FÜR 'LF'
09AE 20 JSR EEA8    EINFÜGEN
09B1 AD LDA 09CC    AKKU ZURÜCKHOLEN
09B4 4C JMP EEA8    WEITER NACH OUTTTY

09B7 A9 LDA #A0     VEKTOR LOW DER PRÜFUNG    INITIALISIERUNG
09B9 8D STA A406    DILINK
09BC A8 LDA #09     VEKTOR FÜR HIGH
09BE 8D STA A407    NACH DILINK+1
09C1 A9 LDA #0C     Z.B. FÜR 300 BAUD
09C3 8D STA A417    NACH CNTH30
09C6 A9 LDA #C2
09C8 8D STA A418    NACH CNTL30
09CB 00 BRK         ZURÜCK ZUM MONITOR
```

Die Besonderheiten des USER OUTPUT HANDLER mit indirektem Sprungvektor in Adresse 010A wurden aufgeklärt: Herr Jon Petter Bjerke in Oslo machte als erster darauf aufmerksam, daß das Monitorprogramm in Adresse E9E7 für User ein PLA hätte enthalten müssen, nachdem das auszugebende Zeichen zuvor auf dem Stack abgelegt wurde. Auf die falsche Darstellung hinsichtlich des Carry-Bit bei User im Anwenderhandbuch verwiesen wir bereits in Heft 5 auf Seite 17. Herr Bjerke zeigt folgenden schematischen USER OUTPUT auf:

```
USTART  BCC USINIT
        PLA
        ... NORMALE OUTPUT-ROUTINE
        RTS
USINIT  ... INITIALISIERUNG DER PERIPHERIE
        ...
        RTS
```

Für die Ausgabe von graphischen Zeichen mit dem Programm AIMGRAPH (Heft 6) aus dem Text-Editor mit LIST OUT=U macht Herr Bjerke den nachstehenden Vorschlag, der hier erfolgreich nachvollzogen werden konnte. Damit bieten sich nunmehr auch Möglichkeiten, aus dem Editor heraus in der Schrift anderer Sprachen zu drucken: a) Aus dem Monitorprogramm wird ein Abschnitt von OUTPRI in das RAM übertragen, um eine Änderung vornehmen zu können, und zwar die Bytes $F000-$F044. Die Adressierung des vormals in $F01B beginnenden JSR-Befehls wird in JSR AIGRA (0200 in Heft 6) geändert.b) Man füge eine eine User-Output-Routine wie folgt hinzu

```
OUTGR   BCC INIT
        SEC                    Die Sequenz entspricht OUTA2 im Monitor
        ROR $A411       PRIFLG
        PLA
        PHP
        JSR OUTPRI1     IN DAS RAM ÜBERTRAGENER ABSCHNITT AUS OUTPRI
        PLP
        ROL $A411
        RTS
```

```
INIT   LDA #$0D          INITIALISIERUNGSROUTINE
       JMP $F000
```

c) Die indirekte Sprungadresse in $010a ist auf OUTGR zu richten. Mit Befehl L, OUT=U erhält man dann Abdruck aus dem Text-Editor mit dem neuen Zeichensatz.

Entsprechende Hinweise zum anwerderdefinierten OUTPUT erhielt der Herausgeber auch von Herrn Gebhard Brinkmann in Kaltenengers und von Herrn Peter Janke in Markdorf. Den Einsendern sei für ihre Mitarbeit herzlich gedankt! Ebenso auch Herrn Hans Layer in Ulm. Er weist auf die vielleicht nicht überall bewußt gewordene Tatsache ist, daß man mit dem C-Command des Editors eine fortgesetzte Suche nach einer Zeichenkette vornehmen kann, indem man in einer aufgefundenen Zeile nicht mit RETURN ändert, sondern mit SPACE zu den Folgezeilen fortschreitet. Zu einem Stichwort kann man also eine Referenzliste aufbauen, z.B. auch in Adreßdateien.

PL/65 - HÖHERE PROGRAMMIERSPRACHE FÜR ROCKWELLS SYSTEM 65

Seit etwa November 1978 wird für Rockwells großes Entwicklungssystem der Sprachkompiler PL/65 auf Floppy Disk angeboten (Preis etwa DM 1.000,-). Diese dem ALGOL und einem verkürzten PL1 ähnliche Sprache läßt ein strukturiertes Programmieren mit sehr ausdruckstarken Sprachelementen zu. Der Compiler in 14 kB erzeugt nicht direkt Maschinencode, sondern Assemblercode, der ggfs. nach den Wünschen des Benutzers noch überarbeitet und für Anwendersysteme benutzt werden kann. Dem Vernehmen nach wird auch an einem PASCAL-Compiler gearbeitet.

BUCHBESPRECHUNGEN

David A. Lien "The BASIC Handbook - An Encyclopedia of the BASIC Computer Language", Compusoft Publishing, San Diego. 2. Aufl. 1979, 360 S.

In diesem umfassenden Nachschlagewerk werden über 250 Kommandos, Anweisungen und Funktionen der Sprache BASIC dargestellt. Dabei sind die Sprachimplementierungen von etwa 50 Computern berücksichtigt, nicht nur diejenigen von Mikroprozessorsystemen, sondern auch die von bekannteren MDT-Computern. In sauberer grafischer Darstellung wird die Auswirkung jedes Befehles erläutert. Ein Testprogramm dafür und die Ergebnisse eines Probelaufes schließen sich an. Es folgen Hinweise auf mögliche weitere Anwendungen des Sprachelements und auf mögliche Ersatzbefehle. Damit ist es möglich, Programmvorlagen, die für andere Interpreter geschrieben sind, zu analysieren und für das eigene Gerät anzupassen. Für die 65xx-Mikros sind die Interpreter aller Computer außer AIM 65/PC100 und SYM-1 berücksichtigt. Das Microsoft-BASIC der ersteren bedeutet gegenüber dem Inhalt des Buches jedoch keinerlei zusätzlichen Dialekt. - Bezug über Micro-Shop-Bodensee, M & R Nedela, ca. DM 39,-.

C. Lorenz "Programmierhandbuch für PET", Hofacker-Verlag, München 1979, 325 S., DM 29,80.

Eines jener enttäuschenden Bücher, die etwa zur Hälfte Datenblätter der Lieferfirmen blind nachdrucken bzw. sich bemühen, weitere Produkte des Hofacker-Verlages anzupreisen. Dem Autor gelingt es nicht, einen Zusammenhang in seine Darstellungen zu bringen und den unvorbereiteten Leser wirklich zu belehren. Fehler und geringe sprachliche Disziplin treten hinzu.

Der PET - Assembler

In Heft 6 auf Seite 29 wurde der von Herrn K. Lehner verfaßte PET-Assembler abgedruckt. Dieser Beitrag fand wegen seiner Nützlichkeit eine sehr zustimmende Aufnahme und führte zu weiteren Anregungen.

Zunächst sind einige Berichtigungen anzubringen: Eine Variable DI gibt es nicht, jedoch ein DI. Daher muß es richtig heißen:

```
5405 ABS(DI)
5420 POKE PC,DI.
```

Am Ende der Zeilen 28100-28130 ist jeweils :RETURN nachzutragen. In Zeile 520 ist CLC,24, versehentlich doppelt abgedruckt worden.

Nun zu den Verbesserungen: Herr Peter Trübger aus Hamburg störte sich am Kommagebrauch in Anführungszeichen und ersann daher eine Möglichkeit, mit INPUT eine Zeile einzugeben, die auch Kommas enthalten darf, ohne daß durch sie die Einlesung in die Variable abgebrochen wird. Er eröffnet die Tastatur in 1010 für INPUT# und prüft dann den Input-Buffer:

```
1010 OPEN 1,0,1:INPUT#1,A$:FOR X=10 TO 50:IF PEEK(X)=0 THEN L=X-1:X=50
1011 NEXT:A$="": FOR X=10 TO L: A$=A$+CHR$(PEEKK(X)): NEXT: CLOSE1
1012 PRINT: RESTORE: R$=""
```

Die Zeile 40 entfällt damit als Hinweis. Man kann jetzt wie üblich z.B. eintippen: LDA ($EA),Y

Eine weitere Bearbeitung wurde vom Herausgeber vorgenommen: Zunächst einmal wird in 1002 eine laufende Anschreibung des Zuordnungszählers PC (*=) vorgenommen, zusammen mit der Zahl der noch freien Bytes, berechnet bis zum Ende des Tape Buffer 2 bzw. bis zum Ende des Hauptspeichers.

In die Zeilen 50-87 wurde eine interaktive Abfrage nach der gewünschten Startadresse des Maschinenprogrammes gelegt. Diese liegt entweder im Puffer des zweiten Magnetbandes (826) oder an anderer vom Benutzer bestimmter Stelle oben im RAM. PC wird dann automatisch zugeordnet, der dem BASIC zur Verfügung stehende Speicherraum entsprechend beschränkt.

```
50 PRINT "ASSIGN BUFFER": INPUT W$
60 IF W$="Y" THEN PC=826: RE=1008:GOTO 87
70 IF W$ ><"N" GOTO 50
80 PRINT"STARTADRESSE=?": INPUT PC: RE=8193
85 IF PC <7198 THEN PRINT"ACHTUNG"
87 WH=INT(PC/256): WL=PC-256*WH: AN=PC
```

Am Ende der Assemblierung wurde in 8130-8176 eine Möglichkeit geschaffen, das erzeugte Maschinenprogramm sofort als DATAFILE (mit Namen) auf Cassette zu schreiben. Anfangs- und Endadresse werden von AN bzw. PC-1 direkt beigesteuert. Die Besonderheiten des PRINT#-Befehles und seines Gegenstückes, des INPUT# wurden berücksichtigt, ebenso der notwendige Abstand zwischen Bandsätzen im DATAFILE.

```
8130 PRINT"DUMP DES MASCHINENPROGRAMMES?": INPUT W$: IF W$="N" THEN END
8135 IF W$ ><"Y" THEN 8130
8140 PRINT"FILENAME IS ?": INPUT W$
8150 OPEN 1,1,1,W$
8151 BU=0
8160 FOR Q=AN TO PC-1
8162 PRINT#1,PEEK(Q): PRINT#1,CHR$(13)
8163 BU=BU+2: IF BU<192 THEN 8174
```

```
8164 T=TI
8166 POKE 59411,53:IF (T-TI)<20 THEN 8166
8168 POKE 59411,61
8170 BU=BU-192
8174 NEXT Q
8176 CLOSE1:END
```

Zum Wiederlesen von DATAFILES(Maschinenprogrammen) wurde der nachfolgende DATALOADER entwickelt, der recht vielseitig eingesetzt werden kann:

```
63000 REM DATALOADER
63001 PRINT "[SHIFT-CLR]"
63010 PRINT"DATA INPUT FROM TAPE 1"
63020 PRINT"AUSFUEHREN?": INPUT G$
63030 IF G$="N" THEN END
63040 PRINT"ANFANG=?": INPUT AN
63050 PRINT "ENDE=?": INPUT EN
63060 PRINT "NAME=?": INPUT F$
63070 OPEN 1,1,0,F$
63080 FOR I=AN TO EN
63090 INPUT#1,X: POKE I,X
63100 PRINT I,X
63110 NEXT I
63120 CLOSE1
```

Hinsichtlich des PET-Assemblers bleiben noch kleine Schönheitsfehler. Mit wenigen Abänderungen kann man vermeiden, daß ein unschädlicher Bedienungsfehler zum Abbruch des Assemblerprogrammes und damit zum Verlust der Symboltabelle führt. Ungelöst ist offensichtlich noch der Sprung mit 'JMP' zu einer Adresse in der Zeropage.

R.L.

****

PET VIDEO DRIVER

Ing. (grad.) Uwe Kornnagel, Lahnstraße 6, D-6096 Raunheim/Main

E: *A set of utilities in machine language allows a multitude of screen manipulations for display of rows and columns with characters by choice, coordinates or a frame.*

Der Autor legt eine Reihe sehr nützlicher Routinen in Maschinensprache vor, die der Video-Ausgabe des PET2001 dienen: Der Cursor kann adressiert werden, Zeilen und Spalten werden mit Zeichen nach Wahl des Betreibers gefüllt oder es wird ein Achsenkreuz gelegt. Diese utilities wurden beim Herausgeber erfolgreich nachvollzogen. Eine Fülle von Anwendungsmöglichkeiten ist gegeben, nicht nur für die Darstellung von Meßwerten oder geometrischen Punkten, sondern auch für die weitere Bildschirmgraphik. Es liegt in der Phantasie des Anwenders, ein übergeordnetes BASIC-Programm zu entwickeln, das den Bildschirm nach seinen Wünschen nutzt. Die Lesergemeinde sei ermuntert, interessierende Anwendungen mitzuteilen.

Bei der Programmeingabe bediene man sich des PET-Assemblers aus Heft 6 mit obenstehenden Nachträgen. - Der Autor weist in anderem Zusammenhang darauf hin, daß für den PET2001 entwickelte Programmcassetten des Hofacker-Verlages nach seinen Beobachtungen nicht auf den Geräten der Serie 3000 laufen. (R.L.)

DAS ARBEITEN MIT VIDEODRIVER 2:

1.) Cursor setzen.

POKE 245,Zeile(0-24): POKE 226,Spalte(0-39): SYS(826): PRINT.....

Beispiel: Das WORT "PET" soll in die 10. Zeile und 20. Spalte geschrieben werden.

POKE 245,10:POKE226,20:SYS(826):?"PET"

2.) Cursor-setzen mit Zeilenlöschung.

POKE 245,Zeile:POKE 226,Spalt:SYS(872)

3.) Waagerechten Strich in eine beliebige Zeile.

POKE 245,Zeile:SYS(876)

4.) Zeile mit einen beliebigen Chr. füllen.

POKE 245,Zeile: POKE 1023,Chr. SYS(880)

5.) Zeile in RVS - Darstellung.

POKE 245,Zeile: SYS(900)

Beispiel: Die Zeile 10 soll RVS - dargestellt werden.

POKE 245,10:SYS(900)

6.) SPALTE in RVS - Darstellung.

POKE 226,Spalte :SYS(915)

7.) Bildschirm in RVS - Darstellung.

SYS(936)

8.) Spalte löschen.

POKE 226,Spalte:SYS(949)

9.) Senkrechter Strich in eine beliebige Spalte.

POKE 226,Spalte:SYS(953)

10.) Spalte mit einen bel. Chr. füllen.

POKE 1023,*chr*:POKE 226,SPALTE: SYS(957)

11.) Das Achsenkreuz an bel. Stelle des Bildschrims.

POKE 245,Zeile:POKE 226,Spalte: SYS(983)

Beispiel: Das Achsenkreuz soll seinen Ursprung in der 12. Zeile und der 15. Spalte haben.

POKE 245,12:POKE 226,15:SYS(983)

Beispiel zum Zeichnen eines Rahmens: POKE 1023,102

```
10 ?"[S] ";:SYS 880:SYS(957)
20 POKE245,24:POKE226,39:SYS(880):SYS(957)
30 WAIT 525,1:GET T$:END
```

([S] = 'HOME')

```
0826    LINE    =245
        CLMN    =226
        VIDP    =224
        CHR0    =1023
        CHR1    =50

        ; PROGRAMMSTART
0826    SETCR   LDA #0          ; CURSOR SETZEN
                LDX #128
                LDY LINE
                BEQ END0
        LP1     CLC
                ADC #40
                BCC CT0
                INX
        CT0     DEY
                BNE LP1
        END0    STA VIDP
                STX VIDP+1
                RTS
0848    SVX     LDA LINE        ; VIDEOSTATUS RETTEN
                STA 11
                LDA CLMN
                STA 12
                LDA #24
                STA LINE
                RTS
0861    RVX     LDA 11          ; VIDEOSTATUS RÜCKSETZEN
                STA LINE
                LDA 12
                STA CLMN
                JMP SETCR
0872    CLRLN   LDA #32         ; ZEILE LÖSCHEN UND CURSOR SETZEN
                BNE LINES
0876    XACHS   LDA #'_         ; X-ACHSE IN BEL. ZEILE
                BNE LINES
0880    FILLN   LDA CHR0        ; ZEILE MIT CHR. FÜLLEN
0883    LINES   AND #127        ; ZEILEN-FÜLL-ROUTINE
                STA CHR1
                JSR SETCR
                LDY #39
                LDA CHR1
        LP2     STA (VIDP),Y
                DEY
                BPL LP2
                RTS
```

```
0900  RVSLN  JSR SETCR      ; ZEILE IN RVS - DARSTELLUNG
               LDY # 39
        LP3    LDA (VIDP),Y
               EOR # 128
               STA (VIDP),Y
               DEY
               BPL LP3
               RTS

0915  RVSCN  JSR SVX        ; SPALTE IN RVS - DARSTELLUNG
        LP4    JSR SETCR
               LDY CLMN
               LDA (VIDP),Y
               EOR # 128
               STA (VIDP),Y
               DEC LINE
               BPL LP4
               JMP RVX

0936  RVSSC  JSR SVX        ; BILDSCHIRM IN RVS - DARSTELLUNG
        LP5    JSR RVSLN
               DEC LINE
               BPL LP5
               JMP RVX

0949  CLRCN  LDA # 32       ; SPALTE LÖSCHEN UND CURSOR SETZEN
               BNE COLUM

0953  YACHS  LDA # '1       ; Y - ACHSE IN BEL. SPALTE
               BNE COLUM

0957  FILCN  LDA CHR0       ; SPALTE MIT CHR. FÜLLEN

0960  COLUM  AND # 127       ; SPALTEN - FÜLL - ROUTINE
               STA CHR1
               JSR SVX
        LP6    JSR SETCR
               LDA CHR1
               LDY CLMN
               STA (VIDP),Y
               DEC LINE
               BPL LP6
               JMP RVX

0983  ACHSK  JSR YACHS
               JSR XACHS
               LDA # '+
               JMP $FFD2      ; FFD2 PRINTROUTINE DES PET.

0994  WT0    LDX # 16       ; WAIT MIT BLINKENDEN "?" FÜR GET
        WT1    LDA 514
        WT2    LDY 525
               BNE END1
```

```
                CMP 514
                BEQ WT2
                DEX
                BNE WT1
                LDY # 1
                LDA (VIDP),Y
                EOR # 31
                STA (VIDP),Y
                BNE WT0
1022    END1    RTS
        ,END
```


12.) Wait für GET - Befehl.

SYS 994: GET ....

Beispiel: Es soll der Text :" Eingeben = -E-, Rechnen = -R-" auf dem Bildschirm erscheinen, und E oder R mittels GET eingelesen werden.

```
10 POKE 245,10:POKE 226,5:SYS 872:REM CLR Zeile 11
20 ?"Eingeben = -E-, Rechnen = -R-";:SYS 994:GET T$
30 IF T$ = "E" THEN .......
40 IF T$ = "R" THEN ......
50 GOTO 10
```

In der 11. Zeile erscheint der Text mit einem blinkenden "?".

DEZINAL - DUMP DES PROGRAMMS.

	.6	.7	.8	.9	.0	.1	.2	.3	.4	.5
082.	169	0	162	128	164	245	240	9	24	105
083.	40	144	1	232	136	208	247	133	224	134
084.	225	96	165	245	133	11	165	226	133	12
085.	169	24	133	245	96	165	11	133	245	165
086.	12	133	226	76	58	3	169	32	208	7
087.	169	192	208	3	173	255	3	41	127	133
088.	50	32	58	3	160	39	165	50	145	224
089.	136	16	251	96	32	58	3	160	39	177
090.	224	73	128	145	224	136	16	247	96	32
091.	80	3	32	58	3	164	226	177	224	73
092.	128	145	224	198	245	16	241	76	93	3
093.	32	80	3	32	132	3	198	245	16	249
094.	76	93	3	169	32	208	7	169	221	208
095.	3	173	255	3	41	127	133	50	32	80
096.	3	32	58	3	165	50	164	226	145	224
097.	198	245	16	243	76	93	3	32	185	3
098.	32	108	3	169	219	76	210	255	162	16
099.	173	2	2	172	13	2	208	18	205	2
100.	2	240	246	202	208	240	160	1	177	224
101.	73	31	145	224	208	228	96	xxx	xxx	xxx

Erzeugung „Quasistatistischen Rauschens" durch „Pseudo-Zufallszahlen-Folgen."

Dr. Günther Rüdiger, Elbinger Weg 7, 2057 Reinbek

E: The author scopes quasi-statistical generation of noise with hardware and implements his solutions on a KIM-1, using its ports and signal amplification as described in the User manual.

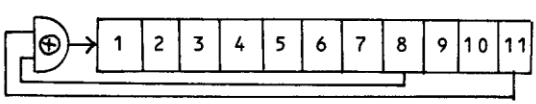
Rauschen, wie z.B. das akustische Rauschen im Bereich der menschlichen Hörfähigkeit, kann mathematisch definiert werden als Überlagerung von allen möglichen Frequenzen eines Spektrums mit einer bestimmten Intensitätsverteilung, die den Charakter des Rauschens bestimmt (Hüllkurve).
Elektrisch-digital kann das akustische Rauschen simuliert werden durch eine statistische Folge von Impulsen mit hoher Folgegeschwindigkeit. Der elektroakustische Wandler (Verstärker-Lautsprecher) macht daraus ein analoges Rauschsignal.
Zur Erzeugung einer „echten" statistischen Impulsfolge stehen verschiedene Methoden zur Verfügung, von denen nur zwei interessante genannt werden sollen:

1. Radioaktiver Zerfall: jeder Zerfall eines Atoms ergibt einen Impuls im Strahlungsdetektor.

2. Roulette: Jedes Wiedererscheinen einer bestimmten Zahl, z.B. der 11, in der unbegrenzten Folge der Spiele eines Tisches, stellt den Impuls dar.

Im ersten Fall können sehr schnelle Impulsfolgen erzeugt werden, deren obere Frequenzgrenze durch die Auflösung des Detektors bestimmt ist. Im zweiten Fall entsteht eine sehr langsame Impulsfolge, die jedoch im Zeittraffer über einen entsprechenden Wandler ebenfalls in ein akustisches Rauschen umgewandelt werden könnte.
Echte statistische Impulsfolgen haben einen Nachteil: die Folgen sind nicht reproduzierbar. Meßtechnische Abläufe lassen sich nicht identisch wiederholen. Dieser sog. Nachteil ist normalerweise natürlich eine unbedingte Forderung an ein statistisches Signal, es darf nicht voraussehbar sein. Bei bestimmten Problemen ist es jedoch günstig, eine quasistatistische (=pseudo-random) Impulsfolge zu erzeugen, die fast alle Eigenschaften einer echten statistischen Folge besitzt, mit einer Ausnahme, sie ist genau reproduzierbar. Um die Erzeugung derartiger Impulsfolgen geht es in diesem Artikel.

Die hardware-mäßige Erzeugung solcher quasistatistischen Impulsfolgen ist lange bekannt und relativ einfach (Literaturstellen 1-4). Grundlage ist ein n-stufiges Schieberegister mit einer Rückführung über ein Exor-Gatter



BEISPIEL: 11 STUFEN, MIT RÜCKFÜHRUNG VON STUFE 8 ⊕ 11

Besser als die zahlreichen Namen für diese Art der Rückkopplung (Exklusiv-ODER-Glied, Modulo-2-Summe, Antivalenz) gibt eine Wahrheitstabelle das Verhalten wieder:

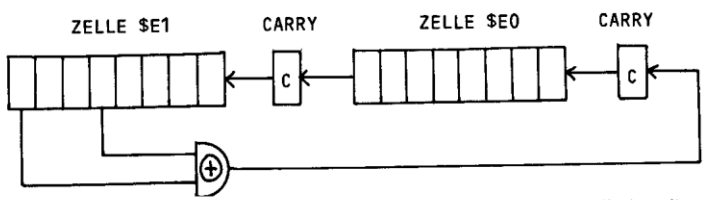
EINGANG 1	EINGANG 2	AUSGANG
0	0	0
0	1	1
1	0	1
1	1	0

Variabel sind die Zahl der Stufen des Schieberegisters und der Rückführungspfad. Je größer die Zahl der Stufen des Registers ist, desto mehr verschiedene Zustände kann es haben, und umso länger ist auch die Identitätsperiode, d.h. die Zahl der aufeinander folgenden verschiedenen Bitmuster bis das Ausgangsmuster wieder erscheint. Die maximal mögliche Anzahl an verschiedenen Bitmustern beträgt (2^n-1), wenn n die Anzahl der Schieberegisterstufen bedeutet. Bezeichnet man die Taktfrequenz des Schieberegisters mit f, so ergibt sich als Identitätsperiode T die Formel:

$$T = \frac{(2^n - 1)}{f} \qquad \text{oder bei } n > 6: \quad \boxed{T \approx \frac{2^n}{f}}$$

mit einem Fehler kleiner 1%.
Bei den meisten Rückführungen wird die maximale Anzahl an Bitmustern allerdings nicht erreicht.

Es gibt nun eine sehr reizvolle Möglichkeit, das Schieberegister mit der antivalenten Rückführung softwaremäßig aufzubauen, und die Funktion, z.B. mit dem KIM 1, ohne zusätzliche Hilfsmittel zu erzeugen. E0 und E1 sind die entsprechenden Speicherplätze in der Z.P. 'C' ist das Carry-Bit. Das Schieberegister mit Rückführung kann dann folgendermaßen aufgebaut werden:



Die Funktion der Rückführung soll an der einfachsten 'Rauschroutine', die bereits das gesamte 'Geheimnis' der Erzeugung quasistatistischer Zufallsimpulsfolgen enthält, erläutert werden:

```
NOISE   A5 E1     RÜCKKOPPLUNGSMASKIERUNG
        29 90
        69 70     ADDITION # ZU $100
        C9 80     EXOR-VERGLEICH
        26 E0
        26 E1
        60
```

Nach Maskierung aller übrigen Bits, außer den Rückführungen (im Beispiel mit 90) und Addition von (100-90)=70 ergeben sich aus dem ursprünglichen Bitmuster folgende vier Möglichkeiten:

MUSTER NACH MASKIERUNG	MUSTER NACH ADDITION	E X O R AUSGANG
1001 0000	0000 0000	0
1000 0000	1111 0000	1
0001 0000	1000 0000	1
0000 0000	0110 0000	0

Es ist also lediglich notwendig, das Muster nach Addition mit der Zahl 80 zu vergleichen, um sofort in C den gewünschten Exor-Ausgang zu erhalten, der dann nur noch weitergeschoben zu werden braucht: Diese etwas trickreiche Methode der Verifizierung der EXOR-Funktion hat 2 Vorteile: Sie ist schnell (die Routine 'NOISE' läuft ohne Rücksprung in 19 μs durch) und sie ist vielfältiger Erweiterungen fähig, die noch erläutert werden sollen. orher sollen aber die einfachsten Anwendungen der Routine 'NOISE' demonstriert werden.

1. Rauschgenerator: PIN PB5 ist als Ausgang zu definieren. Ein Kopfhörer mit einem Innenwiderstand $> 2k\Omega$ ist anzuschließen. Die Routine 'NOISE' soll bei 0220 beginnen.

```
RAUSCHEN   A9 20      LDA #$20
           8D 03 17   STA DDRB
LOC 1      20 20 02   JSR NOISE
           A5 E0      LDA REGØ
           8D 02 17   STA DRB
           38         SEC
           B0 F5      BCS LOC1
```

2. Pseudo-Zufallszahlenerzeugung.

```
           A2 10      LDX #$10
LOC1       20 20 02   JSR NOISE
           CA         DEX
           D0 FA      BNE LOC1
           60         RTS
```

Nach jedem Durchlaufen dieser Routine steht in E0, E1 eine vierstellige Hexadezimalzahl zur Verfügung. 'NOISE' wird dabei 16 mal durchlaufen. Je häufiger, desto unabhängiger werden die Zahlen voneinander. Die gezeigten Abläufe gewinnen natürlich an Geschwindigkeit, wenn man linear programmiert und auf die Unterprogrammsprünge verzichtet.
Die erste Erweiterung: Verlängerung des Schieberegisters in Schritten von jeweils 8 Stufen ist sehr einfach möglich. Ein 8*n stufiges Register wird z.B. durch folgende Modifikation von NOISE verifiziert:

```
NOISE1     A5 80
           8D 02 17
           29 A0
           69 60
           C9 80
           A6 D0
           36 81
           E8
           D0 FB
           F0 EC          IN D0 STEHT DIE HEXAZAHL 100-n
```

Vor dem Anspringen dieser 'Routine ohne Ende' muß ein Pin im B-Port als Ausgang definiert werden. 'n' sei =6. Die Durchlaufzeit beträgt 84 us, also 11 904 Durchläufe pro Sekunde. Das damit aufgebaute Schieberegister hat maximal 2^{48} verschiedene Zustände. Die maximale Identitätsperiode beträgt demnach $2^{48} \times 84 \mu s = 750$ Jahre!

Ob allerdings bei der gewählten Rückführung (Bit 7 und Bit 5 des letzten Byte) diese maximale Periode erreicht wird, bleibt unbestimmt. In jedem Fall ist sie für normale Zwecke als „unendlich" zu bezeichnen. Die Grenzfrequenz beträgt fast 12 KHz.

Die 2. Erweiterung: Variable Rückführung mit automatischem Durchlauf aller möglichen Typen erlaubt die schnelle akustische Überprüfung der Qualität des Rauschens bzw. der pseudostatistischen Signale.
Innerhalb eines Bytes gibt es insgesamt $\frac{8(8-1)}{2} = 28$ verschiedene Rückführungen aus je zwei Bit.

Bezeichnet man mit 'RH' den hexadezimalen Wert des Bits mit der höheren Rückführung, mit 'RL' den der niedrigeren Rückführung, so können die 3 konstanten Zahlenwerte in der Routine 'NOISE' in hexadezimaler Rechnung folgendermaßen berechnet werden:

```
A = RH + RL
B = 100 - RH - RL = 100 - A
C = 100 - RH = B + RL
Beispiel: Rückführung von Bit 7 und Bit 4
          Also: RH = 80 und RL = 0
          Damit ergibt sich:
          A = 90
          B = 70
          C = 80
```

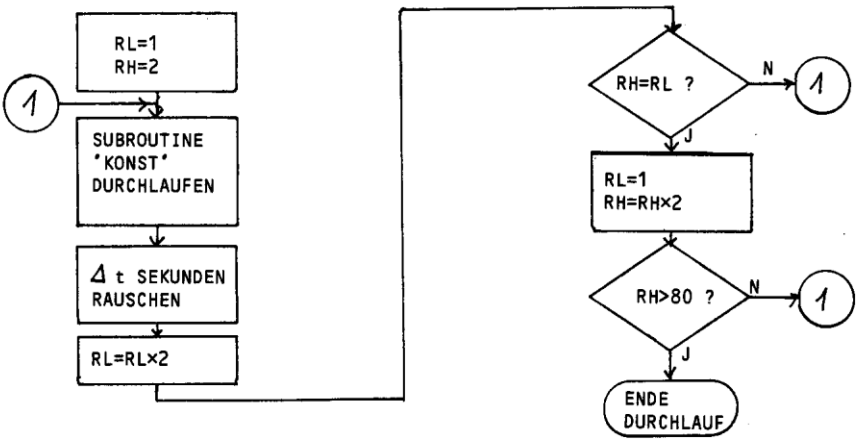
(mit diesen Werten ist die Routine 'NOISE' formuliert).
Diese Rechnungen lassen sich natürlich auch per Programm ausführen, wenn RH und RL vorgegeben sind.

```
Speicherbelegung  RH  in  80
                  RL  in  81
                  A   in  82
                  B   in 83
                  C   in 84

Berechnungsroutine 'KONST'

                  18        CLC
                  A5 80     LDA RH
                  65 81     ADC RL
                  85 82     STA A
                  49 FF     EOR #$FF
                  69 01     ADC #$01
                  85 83     STA B
                  65 81     ADC RL
                  85 84     STA C
                  60        RTS
```

Die 3 Konstanten stehen auf den Speicherplätzen 82-84 zur Verfügung und können von 'NOISE' abgerufen werden durch Modifizierung der folgenden Befehle:

```
29 90                25 82
69 70      IN        65 83
C9 80                C5 84
```

Schließlich besteht noch die Möglichkeit, alle 28 möglichen Rückführungen innerhalb eines Bytes programmgesteuert zu ermitteln und zu durchlaufen. Der dazu notwendige Algorithmus wird im folgenden Flußdiagramm wiedergegeben:



Dieses relativ einfache Flußdiagramm wird nachfolgend als 'gebrauchsfertiges' Programm codiert, wobei die beiden Sub-Routinen 'KONST' und 'NOISE 2' verwendet werden.

```
'28×NOI'  A9 01        LDA #$01
          8D 01 17     STA DRB
          85 81        STA RL
          0A           ASL  (×2)
          85 80        STA RH
          20 XX XX NEXTJSR KONST
          20 XX XX     JSR NOISE2
          06 81        ASL  (RL×2)
          A5 80        LDA RH
          C5 81        CMP RL
          D0 F2        BNE NEXT
          0A           ASL  (×2)
          85 80        STA RH
          A9 01        LDA #$01
          85 81        STA RL
          90 E9        BCC NEXT
          4C 4F 1C     JMP MONITOR

'NOISE2'  A2 04        LDX #$04
          A9 FF    G01 LDA #$FF
          8D 07 17     STA CLK 1:1024
          A5 E2    G02 LDA ST02
          8D 00 17     STA DRA
          25 82        AND A
          65 83        ADC B
          C5 84        CMP C
          26 E0        ROL ST0Ø
          26 E1        ROL ST01
          26 E2        ROL ST02
          2C 05 17     BIT FLAG
          10 EA        BPL G02
          CA           DEX
          D0 E2        BNE G01
          60           RTS
```

Nach Programmstart und Anschluß des Kopfhörers an PIN PA0 durchläuft der Rauschgenerator die 28 verschiedenen Rauschmuster, die z.T. Toncharakter haben und kehrt dann in den normalen KIM-Monitor zurück. Man kann natürlich den Ablauf jederzeit mit 'Reset' unterbrechen und in den Plätzen 80 und 8 in der Z.P. die Art der Rückführung ablesen. Abschließend soll noch darauf hingewiesen werden, daß wenigstens ein Bit der am Aufbau des Schieberegisters beteiligten Speicherplätze eine 1 enthalten muß, weil sonst der Pseudo-Random-Prozeß nicht in Gang kommt. Grund: $0 \oplus 0 = 0$

Literaturhinweise:

1. K. Kock, Elektronik 1968, Heft 3, Seite 69 ff.
2. K.-J. Schmidt, Elektronik 1975, Heft 5, Seite 79 ff.
3. Texas Instruments, „Kochbuch", Seite 168 ff.
4. Anderson et al., Hewlett-Packard-Journal, Sept. 1967, Seite 2 ff.

LITERATURHINWEISE

MICRO 10 (1979), S. 5: M.L. De Jong, ' A Simple 24 Hour Clock for the AIM 65'. Programm mit Einsatz des VIA-Timers T1 im Interruptbetrieb.

MICRO 10 (1979), s. 19: John R. Sherburne 'High Resolution Plotting for the PET'. Verbund von ausführendem Maschinenprogramm (Video-Driver) und aufrufendem BASIC-Programm zur Darstellung komplexer Funktionen, z.B. auch von Lissajous-Figuren.

MICRO 10 (1979):, S. 37: Harvey B. Herman 'A Pet Machine Language Memory Test'. Transciription von Jim Butterfields 'Memory Test' (im First Book of KIM' für den PET.

MICRO 11 (1979), S. 37: Don Rindsberg 'The Ultimate PET Renumber'. Schnelles Programm in Maschinensprache zur Neu-Numerierung von BASIC-Statements.

MICRO 12 (1979), S. 13: 'A PET Hex Dump Program'. BASIC-Programm zur Darstellung hexadezimaler Speicherinhalte.

MICRO 12 (1979), S. 25: J. C. Williams 'A 100 µs 16 Channel A/D Converter for 65xx Microcomputer Systems. Schaltung mit ADC0817 und Programm.

KILOBAUD 4/79, S. 90: Frederick E. Luffman 'Bar Graph Generator'. Darstellung von Balkendiagrammen mit dem PET für statistische Eingabewerte.

KILOBAUD 3/79, S. 62: Gregory Yob 'Pet User Port Cookbook'. Ein Grundsatzartikel zur Nutzung des PET-Anwenderports mit Schaltungen und Programmen. Vorabdruck aus einem Buch, das in diesem Sommer erwartet wird.

BYTE 5/79, S. 52: Bob Haas 'Single Chip Video Controller'. Die Video-Controller Intel 8275, der Motorola MC6845, der NS DP8350 und der SMC 5027 werden im Vergleich vorgestellt. Speziell für den MC6845 folgen Blockdiagramme, ein Schaltungsvorschlag und Betriebssystem-Vorschläge für die 65xx Micros.

BYTE 5/79, S. 130: Chris Tennant 'The Intel 8275 CRT Controller'. Gleichartiger Artikel wie vor mit Beschaltungsvorschlägen.

BYTE 4/79, S. 26: Mark Zimmermann ' Simulating Physical Systems - The Two Dimensional Ideal Gas'. Pet-Programm in BASIC mit Unterprogrammen in Maschinensprache zur Darstellung der Verhältnisse in einem idealen Gas.

6502 User Notes 14 (1979), S. 4: Bob Kurtz 'KIM-1 Disassembler Program'.

6502 User Notes 14 (1979), S. 9: Robert D. Larrabee 'Check OUT'. Ein KIM-Programm zur Programm-Editierung. Mit einstellbarer Geschwindigkeit kann man das Anwenderprogramm zur Anzeige bringen, vorwärts oder rückwärts laufend. Korrekturen können angebracht werden, ferner ist byteweise ein 'Insert' oder Delete' mit Verschiebung des Restprogrammes möglich.

****

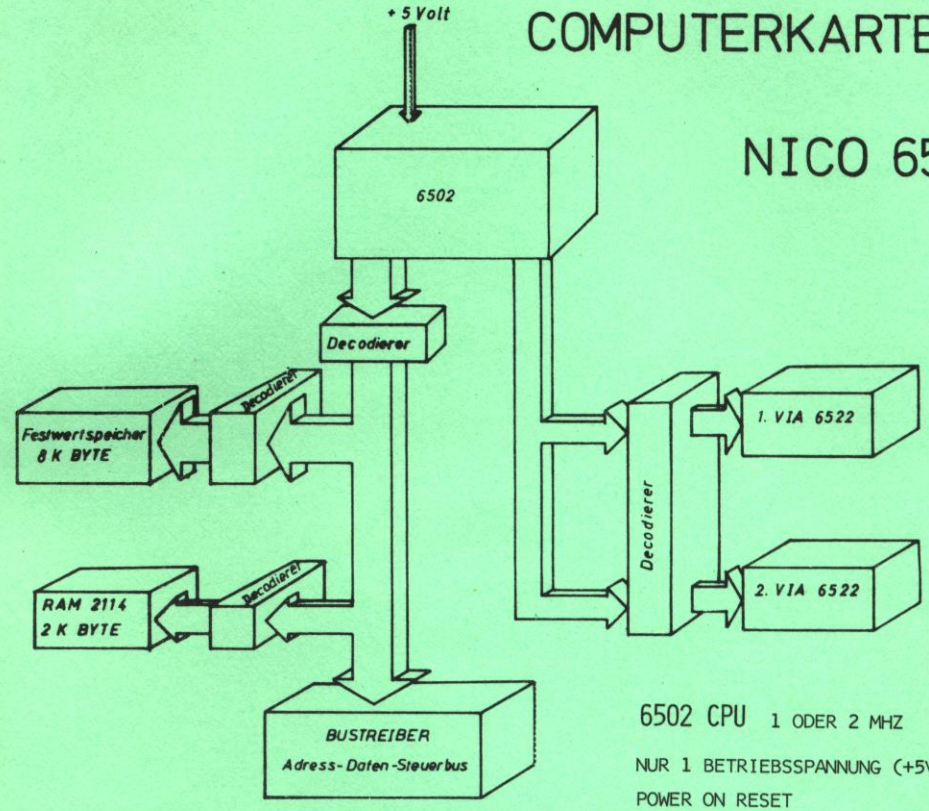
VERLÄNGERN SIE BITTE RECHTZEITIG IHR ABONNEMENT !

Mit diesem 7. Heft erhalten wiederum viele Abonnemten den Hinweis, daß ihr Erstabonnement abgelaufen ist. Geben Sie daher bitte baldmöglichst mit dem Antwortcoupon Nachricht, daß Sie das 65xx MICRO MAG weiter beziehen wollen. Heft 8 erscheint nach der Sommerpause Ende August 1979. Veranlassen Sie dann bitte Scheckzahlung oder Überweisung.

****

COMPUTERKARTE

NICO 65



6502 CPU 1 ODER 2 MHZ

NUR 1 BETRIEBSSPANNUNG (+5V)

POWER ON RESET

8 K EPROM mit Anwenderprogramm, Adreßbereich über Lötbrücke festlegbar

2 K RAM, statisch

2 INTERFACEBAUSTEINE 6522 VIA mit 40 I/O-Leitungen, 4 Timern, 2 Zählern, Handshake-Betrieb, 2 Schieberegistern, 7 Interruptmöglichkeiten

TREIBER FÜR ADRESS-, DATEN- UND STEUERBUS

VG64 STECKERLEISTE für Erweiterungen

Leiterbahnen glanzverzinnt, Stecksockel. Expandierbar auf gesamten Prozessorbereich, gleich 64 k.

PREIS: DM 390,- +MWST, in Grundausstattung bestückt

ALS ERWEITERUNG SIND LIEFERBAR:

Combo-Karte mit 8 k RAM und 2 Stück 6522 VIA (siehe oben),
TV-Interfacekarte mit 2 k Bildschirmspeicher in Vorbereitung
8 Bit A/D Wandler
8 Bit D/A Wandler

I.DOHMANN

ELEKTRONISCHE BAUGRUPPEN
IM WIEHAGEN 15
4830 GÜTERSLOH 12
TEL.: 05241 - 67 480

65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

HERAUSGEBER:
DIPL.-VOLKSWIRT ROLAND LÖHR
HANSDORFER STRASSE 4
2070 AHRENSBURG
☎ (04102) 55 816

65xx MICRO MAG erscheint zweimonatlich. Beiträge, die nicht besonders gekennzeichnet sind, stammen vom Herausgeber.

KATHODENSTRAHLSATZ: DINGES + FRICK, WIESBADEN. OFFSETDRUCK: DRUCKARTIST GERHARD M. MEIER, HAMBURG.

BEZUGSBEDINGUNGEN: Abonnement für 6 Hefte im Inland DM 40,- (Endpreis). Ausland/Foreign: DM 46,- (surface mail). Firmen erhalten Rechnung. Rechnungserteilung sonst nur auf Wunsch. Richten Sie bitte Ihre Überweisung/Euroscheck an:

Roland Löhr, Konto 20/01121, Vereins- und Westbank, BLZ 200 300 00.

Alle früher erschienenen Hefte dieser Zeitschrift sind weiterhin lieferbar. Einzelne Hefte können zu DM 7,- inkl. Porto nachbezogen werden.

REDAKTIONS- UND ANZEIGENSCHLUSS FÜR NR. 8 ⇨ 14. AUG. 79

****